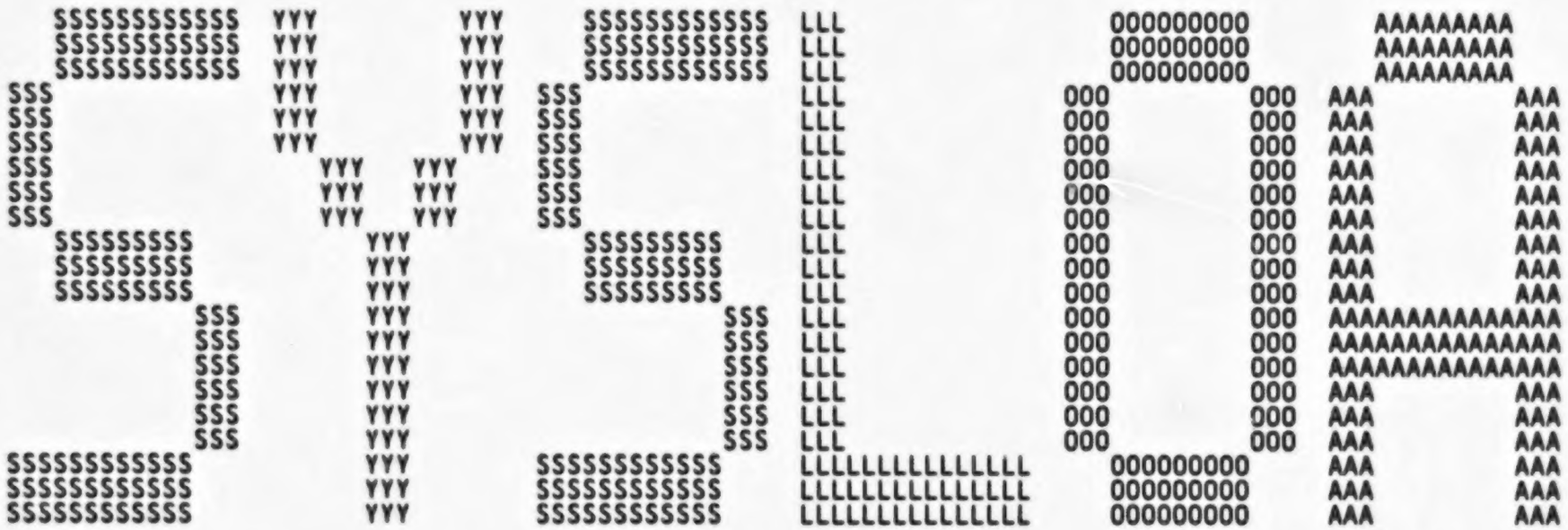
SSSSSSSSSSS YYY YYY SSSSSSSSSSS LLL OOOOOOOOO AAAAAAAAA
SSSSSSSSSSS YYY YYY SSSSSSSSSSS LLL OOOOOOOOO AAAAAAAAA
SSSSSSSSSSS YYY YYY SSSSSSSSSSS LLL OOOOOOOOO AAAAAAAAA
SSS YYY YYY SSS LLL OOO OOO AAA AAA
SSS YYY YYY SSS LLL OOO OOO AAA AAA
SSS YYY YYY SSS LLL OOO OOO AAA AAA
SSS YYY YYY SSS LLL OOO OOO AAA AAA
SSS YYY YYY SSS LLL OOO OOO AAA AAA
SSS YYY YYY SSS LLL OOO OOO AAA AAA
SSSSSSSSS YYY SSSSSSSSS LLL OOO OOO AAA AAA
SSSSSSSSS YYY SSSSSSSSS LLL OOO OOO AAA AAA
SSSSSSSSS YYY SSSSSSSSS LLL OOO OOO AAA AAA
SSS YYY SSS LLL OOO OOO AAAAAAAAAAAAAAA
SSS YYY SSS LLL OOO OOO AAAAAAAAAAAAAAA
SSS YYY SSS LLL OOO OOO AAAAAAAAAAAAAAA
SSS YYY SSS LLL OOO OOO AAA AAA
SSS YYY SSS LLL OOO OOO AAA AAA
SSSSSSSSSSS YYY SSSSSSSSSSS LLLLLLLLLLLLLLL OOOOOOOOO AAA AAA
SSSSSSSSSSS YYY SSSSSSSSSSS LLLLLLLLLLLLLLL OOOOOOOOO AAA AAA
SSSSSSSSSSS YYY SSSSSSSSSSS LLLLLLLLLLLLLLL OOOOOOOOO AAA AAA

FILEID**CSPCALL

```
  CCCCCCCC    SSSSSSSS  PPPPPPPP      CCCCCCCC    AAAAAA    LL          LL
  CCCCCCCC    SSSSSSSS  PPPPPPPP      CCCCCCCC    AAAAAA    LL          LL
CC            SS        PP      PP  CC          AA    AA  LL          LL
CC            SS        PP      PP  CC          AA    AA  LL          LL
CC            SS        PP      PP  CC          AA    AA  LL          LL
CC            SS        PP      PP  CC          AA    AA  LL          LL
CC              SSSSSS  PPPPPPPP    CC          AA    AA  LL          LL
CC              SSSSSS  PPPPPPPP    CC          AA    AA  LL          LL
CC                    SS PP        CC          AAAAAAAAAA  LL          LL
CC                    SS PP        CC          AAAAAAAAAA  LL          LL
CC                    SS PP        CC          AA    AA  LL          LL          ....
CC                    SS PP        CC          AA    AA  LL          LL          ....
  CCCCCCCC  SSSSSSSS    PP          CCCCCCCC  AA    AA  LLLLLLLLLL  LLLLLLLLLL  ....
  CCCCCCCC  SSSSSSSS    PP          CCCCCCCC  AA    AA  LLLLLLLLLL  LLLLLLLLLL  ....

LL              IIIIII    SSSSSSSS
LL              IIIIII    SSSSSSSS
LL                II    SS
LL                II    SS
LL                II    SS
LL                II    SS
LL                II      SSSSSS
LL                II      SSSSSS
LL                II            SS
LL                II            SS
LL                II            SS
LL                II            SS
LLLLLLLLLL      IIIIII    SSSSSSSS
LLLLLLLLLL      IIIIII    SSSSSSSS
```

Table of contents

(11) 405 'CSP$INIT - Init CSP data structures upon load'
(12) 437 'CLEAN_UP - ACKMSG Rcv cleanup routine'
(13) 475 'CSP$DISPATCH - Dispatch on received ACKMSG message'
(14) 589 'EXE$CSP_COMMAND Receive commnad from CSP process'
(15) 757 'EXE$CSP_BRDCST - Send CSP request to all nodes'
(16) 983 'EXE$ALLOC_CSD - Allocate and initialize a CSD block'
(17) 1131 'EXE$DEALLOC_CSD Deallocate CSD or mark it for deletion'
(18) 1171 'EXE$CSP_CALL - Send a request message to local or remote CSP'
(20) 1320 'KAST - Special Kernel AST entry point'
(20) 1321 'AST - Normal Kernel AST entry point'
(21) 1374 'PROC_EVENT_ASY - Process CSD event if process is still around'
(21) 1375 'PROC_EVENT - Process CSD event'
(22) 1465 'ACT_INSQUE - Queue ACB to CSP$Q_ACB_IDLE'
(22) 1466 'ACT_REMQUE - Remove ACB from current (internal) queue'
(23) 1500 'ACT_GET_CDRP - Allocate a warm CDRP for block transfer'
(24) 1567 'ACT_FORK_WAIT - Fork and wait for up to 1 second'
(25) 1621 'ACT_REQ_ILL_BT - Request illegal block-transfer'
(25) 1622 'ACT_BLOCK_XFER - Request ACKMSG Block Transfer'
(26) 1740 'ACT_NO_AST - No AST to deliver - deallocate CSD if broadcast'
(26) 1741 'ACT_GIVE_UP - Retry count has be exhausted, give up'
(26) 1742 'ACT_QUE_KAST - Queue Special Kernel AST to process'
(26) 1743 'ACT_QUE_AST - Queue Normal Kernel AST to process'
(27) 1798 'ACT_SYN_ERROR - Synchronous block transfer error'
(28) 1826 'ACT_REQ_DEAL - Illegal user deallocation request'
(29) 1879 'ACT_DEALL - Deallocate CSD, return quotas'
(30) 1933 'ACT_BUG - Bugcheck failure'
(30) 1934 'ACT_NYI - Not-yet-implemented error'
(30) 1935 'ACT_NOP - No-operation'

```
0000     1          .TITLE  CSPCALL        - Loadable Exec support for CSP
0000     2          .IDENT  'V04-000'
0000     3
0000     4  ;*****************************************************************************
0000     5  ;*                                                                           *
0000     6  ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
0000     7  ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0000     8  ;*  ALL RIGHTS RESERVED.                                                     *
0000     9  ;*                                                                           *
0000    10  ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED     *
0000    11  ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE    *
0000    12  ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER     *
0000    13  ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY     *
0000    14  ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY     *
0000    15  ;*  TRANSFERRED.                                                             *
0000    16  ;*                                                                           *
0000    17  ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE     *
0000    18  ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
0000    19  ;*  CORPORATION.                                                             *
0000    20  ;*                                                                           *
0000    21  ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS     *
0000    22  ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
0000    23  ;*                                                                           *
0000    24  ;*                                                                           *
0000    25  ;*****************************************************************************
0000    26  ;
0000    27  ;++
0000    28  ;
0000    29  ; FACILITY:     VMS
0000    30  ;
0000    31  ; ABSTRACT:     Routine to call the Cluster Server Process on another node.
0000    32  ;
0000    33  ; AUTHOR:       Paul R. Beck
0000    34  ;
0000    35  ; DATE:         21-MAR-1983
0000    36  ;
0000    37  ; REVISION HISTORY:
0000    38  ;
0000    39  ;       V03-016 ADE0010         Alan D. Eldridge        18-Jul-1984
0000    40  ;               Consmetic (comments only) cleanup.
0000    41  ;
0000    42  ;       V03-015 ADE0008         Alan D. Eldridge        24-May-1984
0000    43  ;               Add bug-checks to avoid pool corruption when deallocating
0000    44  ;               packets.  This has proven to be a problem area.
0000    45  ;
0000    46  ;       V03-014 ADE0008         Alan D. Eldridge        22-May-1984
0000    47  ;               Bias ACB$W_WAIT_CNT in EXE$CSP_BRDCST while the routine is
0000    48  ;               referencing the master ACB copy.  This is needed since the code
0000    49  ;               is a referencer -- race conditions could otherwise cause the
0000    50  ;               ACB$V_STS_WAIT flag to be cleared prematurely by DEALL_CSD.
0000    51  ;
0000    52  ;       V03-013 ADE0007         Alan D. Eldridge        18-May-1984
0000    53  ;               Clear parent pointer in offspring ACB when deallocating
0000    54  ;               offspring.  It was being deallocated in the parent ACB.
0000    55  ;
0000    56  ;       V03-011 ADE0006         Alan D. Eldridge        26-Apr-1984
0000    57  ;               Erase ACB$V_WAIT at end of EXE$CSP_BRDCST if ACB$W_WAIT_CNT
```

```
0000     58 ;                  is zero.
0000     59 ;
0000     60 ;       V03-010 ADE0005         Alan D. Eldridge        12-Apr-1984
0000     61 ;                Make default retry count 4 -- it was 30.
0000     62 ;
0000     63 ;       V03-010 ADE0004         Alan D. Eldridge        22-Mar-1984
0000     64 ;                Fix EXE$CSP_COMMAND handling of CSP$_LOCAL request.
0000     65 ;
0000     66 ;       V03-009 DWT0193         David W. Thiel          15-MAR-1984
0000     67 ;                Change interface to ACKMSG block transfer.
0000     68 ;
0000     69 ;       V03-008 ADE0003         Alan D. Eldridge        28-Feb-1984
0000     70 ;                Add support for CSP$_LOCAL call in EXE$CSP_COMMAND.
0000     71 ;
0000     72 ;       V03-007 ADE0002         Alan D. Eldridge        6-Feb-1984
0000     73 ;                Move CSD address to R2 in EXE$CSP_BRDCST before call to WAIT.
0000     74 ;                Call scheduler at IPL$_SYNCH.  Check ACB$W_WAIT_CNT before
0000     75 ;                clear ACB$V_STS_WAIT.
0000     76 ;
0000     77 ;       V03-006 ADE0001         Alan D. Eldridge        9-Dec-1983
0000     78 ;                Rewrite to use the ACKMSG of the Connection Manager rather
0000     79 ;                than DECnet.  Merge module CSPALLOC into this one in order
0000     80 ;                keep all special buffering details local to one module.
0000     81 ;                Add state table, etc.
0000     82 ;
0000     83 ;       V03-005 JLV0309         Jake VanNoy             5-OCT-1983
0000     84 ;                Check status after call to EXE$ALLOC_CSD.
0000     85 ;
0000     86 ;       V03-004 JLV0305         Jake VanNoy             29-AUG-1983
0000     87 ;                Add error checking to EXE$CSP_CALL call in EXE$CSP_BRDCST.
0000     88 ;                Call EXE$DEANONPGDSIZ instead of EXE$DEANONPAGED.
0000     89 ;
0000     90 ;       V03-003 PRB0231         Paul R. Beck    13-JUL-1983 21:33
0000     91 ;                Fix bugs in broadcast.
0000     92 ;                Change "empty slot" test in main routine.
0000     93 ;
0000     94 ;       V03-002 PRB0203         Paul R. Beck    7-JUN-1983 22:53
0000     95 ;                Fix non-PIC definition of NET0:
0000     96 ;                Add broadcast capability.
0000     97 ;
0000     98 ;       V03-001 PRB0164         Paul R. Beck    22-APR-1983 14:28:31
0000     99 ;                Add PSECT.
0000    100 ;--
```

```
0000    102 ;+
0000    103 ;
0000    104 ;   Future enhancements:
0000    105 ;
0000    106 ;   1.  Create a better bug-check code.  INCONSTATE is temporary.
0000    107 ;
0000    108 ;   2.  Do a better job about image rundown.
0000    109 ;
0000    110 ;   3.  What happens if a user tries to ^Y-Stop in various places (especially
0000    111 ;       after depleting the JIB quota and while in a a wait state allocating
0000    112 ;       memory).
0000    113 ;-
0000    114
0000    115 ;
0000    116 ;  Definitions
0000    117 ;
0000    118             $ACBDEF
0000    119             $CSBDEF
0000    120             $CSDDEF
0000    121             $CSPDEF
0000    122             $CDRPDEF
0000    123             $CLSMSGDEF
0000    124             $CLUBDEF
0000    125             $CLUBTXDEF
0000    126             $DYNDEF
0000    127             $FKBDEF
0000    128             $IPLDEF
0000    129             $JIBDEF
0000    130             $PCBDEF
0000    131             $PHDDEF
0000    132             $PRIDEF
0000    133             $RSNDEF
0000    134             $SBODEF
0000    135             $SSDEF
0000    136             $VADEF
0000    137
0000    138
0000    139
```

```
                     0000   141 ;
                     0000   142 ;   Macro to setup up a routine dispatch table
                     0000   143 ;
                     0000   144 .MACRO  $DSP_TABLE  list                          ; Setup dispatch table
                     0000   145
                     0000   146         .MACRO  $dspent _$dspinx,_$dspact
                     0000   147                 .IIF GT,  <_$dspinx- $maxinx>,   $maxinx = _$dspinx
                     0000   148                 .       = _$tmp + <4 * _$dspinx>
                     0000   149                 .long   _$dspact - _$tmp
                     0000   150         .ENDM   $dspent
                     0000   151
                     0000   152         _$tmp    = .
                     0000   153         _$maxinx = 0
                     0000   154         .IRP    a,<LIST>
                     0000   155                 $dspent a
                     0000   156         .ENDR
                     0000   157
                     0000   158         . = _$tmp + <4 * _$maxinx> + 4
                     0000   159 .ENDM   $DSP_TABLE
                     0000   160
                     0000   161 ;
                     0000   162 ;   Macro to create and fill the event state table.
                     0000   163 ;
00000006             0000   164 CEV$K_STATES  =  6                               ; Number of columns in the table
FFFFFFFF             0000   165 CEV$_MAX_EVT  = -1                               ; Init the number of rows
00000000             0000   166 CEV$_EXIT     =  0                               ; Define termination event
                     0000   167
                     0000   168 .MACRO  $CEV  event, i,f,x,k,a,s                 ; Create state table entries
                     0000   169                                                  ; for the specified event
                     0000   170         CEV$_MAX_EVT = CEV$_MAX_EVT + 1         ; Bump max event value
                     0000   171         CEV$_'event' = CEV$_MAX_EVT             ; Define circuit event symbol
                     0000   172
                     0000   173                 $ENT    i,_i                    ; Create table entry
                     0000   174                 $ENT    f,_f
                     0000   175                 $ENT    x,_x
                     0000   176                 $ENT    k,_k
                     0000   177                 $ENT    a,_a
                     0000   178                 $ENT    s,_s
                     0000   179 .ENDM   $CEV
                     0000   180
                     0000   181 .MACRO  $ENT    entry,def_sta                   ; Create state table entry
                     0000   182
                     0000   183                 _$ent = %LENGTH(entry)-1
                     0000   184                 CEV$K_sta_. = CEV$K_sta'def_sta'; Define default next state
                     0000   185
                     0000   186         .IF IDN,entry,?                         ; ? => bug
                     0000   187             .BYTE  CEV$K_sta_.                  ; Use current state
                     0000   188             .BYTE  2                            ; Action is bug-check
                     0000   189         .IFF
                     0000   190             .BYTE  CEV$K_sta_%EXTRACT(0,1,entry); Setup next state
                     0000   191             .BYTE  %EXTRACT(1,_$ent,entry)      ; Setup action routine index
                     0000   192         .ENDC
                     0000   193 .ENDM   $ENT
                     0000   194
                     0000   195
```

```
                        0000    197
                        0000    198 .MACRO  $RSP_CEV_TAB, LIST                      ; CSPMSG$K_RSP to CEV$_ mapping
                        0000    199
                        0000    200         .MACRO  $make_entry, rsp, cev
                        0000    201                 . = _$START + cspmsg$k_rsp_'rsp'
                        0000    202                 .byte cev$_'cev'
                        0000    203         .ENDM   $make_entry
                        0000    204
                        0000    205         _$start = .
                        0000    206                 .byte   0 [cspmsg$k_rsp_max+1]          ; Init table
                        0000    207         _$end   = .
                        0000    208
                        0000    209         .IRP    member,<list>                       ; Fill table
                        0000    210                 $make_entry member
                        0000    211         .ENDR
                        0000    212         .=_$end
                        0000    213
                        0000    214 .ENDM   $RSP_CEV_TAB
                        0000    215
```

```
                     0000   217 ;
                     0000   218 ;  Define CLSMSG format
                     0000   219 ;
                     0000   220 $DEFINI CSPMSG
                     0000   221 $EQULST CSPMSG$K_RSP_,,0,1,-                 ; Define response codes
                     0000   222 <-
                     0000   223         <NOP>,-                              ; Should never be used
                     0000   224         <ILL>,-                              ; Illegal CSPMSG$K_RSP_xx code specified
                     0000   225         <BUSY>,-                             ; Remote CSP is busy, try later
                     0000   226         <NOCSP>,-                            ; No CSP process
                     0000   227         <RO>,-                               ; Read/only completion
                     0000   228         <RW>,-                               ; Read/write completion
                     0000   229         <BADCSD>,-                           ; Illegal CSD detected
                     0000   230         <ASYNERR>,-                          ; Asynchronous block transfer failure
                     0000   231         <SYNERR>,-                           ; Synchronous block transfer failure
                     0000   232         <MAX>,-                              ; Not a legal response code -- used
                     0000   233 >                                            ; to mark end of list
                     0000   234
00000018             0000   235 .= CLMHDR$K_BT_LENGTH                        ; Skip over ACKMSG header
                     0018   236
                     0018   237 $DEF     CSPMSG$B_RSP       .BLKB 1       ; Response code
                     0019   238 $DEF     CSPMSG$B_SPARE     .BLKB 1       ; Reserved -- used here for alignment
                     001A   239 $DEF     CSPMSG$W_CLIENT    .BLKW 1       ; Client i.d.
                     001C   240 $DEF     CSPMSG$L_CSD_SIZE  .BLKL 1       ; Size of CSD
00000020             0020   241         CSPMSG$K_LENGTH = .                ;
                     0020   242 $DEFEND CSPMSG                              ;
                     0000   243
                     0000   244 $DEFINI ACB                                 ; Define our own ACB extensions
                     0000   245                                             ;
00000020             0000   246 .= <ACB$K_LENGTH + 15>  & ^C<15>             ; Goto end of normal ACB honoring normal
                     0020   247                                             ; pool granularty
                     0020   248         ;
                     0020   249         ;  A copy of the AST and PID are needed in the ACB to prevent a block
                     0020   250         ;  tranfer or a client from corrupting the ones in the CSD.
                     0020   251         ;
                     0020   252 $DEF     ACB$L_USER_AST   .BLKL   1        ; User's AST address
                     0024   253 $DEF     ACB$L_USER_PID   .BLKL   1        ; User's PID
                     0028   254 $DEF     ACB$W_WAIT_CNT   .BLKW   1        ; Used if ACB$V_STS_BCST is set
                     002A   255                                            ; -- # of outstanding broadcasts
                     002A   256 $DEF     ACB$W_LAST_INX   .BLKW   1        ; -  Last CSB index used
                     002C   257 $DEF     ACB$L_PARENT     .BLKL   1        ; Used if ACB$V_STS_BCST is clear
                     0030   258                                            ; -- 0 means no parent
                     0030   259 $DEF     ACB$B_STA        .BLKB   1        ; CEV$K_STA_xxx code used by state table
                     0031   260 $DEF     ACB$B_STS        .BLKB   1        ; The following:
                     0032   261                                            ;
                     0032   262         $VIELD  ACB,0,-                    ;
                     0032   263                         <<STS_ASY,,M>  -;  Used to determine if return was async
                     0032   264                         ,<STS_QUE,,M>  -;  Set if ACB queue header is in use
                     0032   265                         ,<STS_WAIT,,M> -;  While set, don't return to user
                     0032   266                         ,<STS_BCST,,M> -;  Set if part of broadcast
                     0032   267                         ,<STS_PCNT,,M> -;  Set if part of parent's WAIT_CNT
                     0032   268                         >                  ;
                     0032   269 $DEF     ACB$W_RETRY      .BLKW   1        ; Retries allowed (signed value)
00000004             0034   270         ACB$K_RETRY      = 4              ; Max number of retries allowed
00000034             0034   271         ACB$K_CSPLNG     = .              ; Length of ACB we use
                     0034   272 $DEFEND ACB                                ;
```

```
                                .PSECT  $$$200,NOPIC,EXE,QUAD,RD,WRT

CSP$BEGIN::                                                     ; Starting address for reading
                                                                ; map while debugging
;
; OWN STORAGE:
;


;
;  ACB states
;
$EQULST CEV$K_STA_,,0,1,-
<-
        <I>     -; Initial:     Initial state upon being allocated.
                -;              On the 'idle CSD' queue.
                -;
        <F>     -; Forking:     Waiting 1 sec. before requesting a "warm" CDRP.
                -;              On either some system fork or wait queue.
                -;
        <X>     -; Transfer:    Undergoing block transfer.
                -;              On the 'active transfer' queue.
                -;
        <K>     -; KAST:        In use as a 'special kernel' AST block.
                -;              On the PCB AST queue.
                -;
        <A>     -; AST:         In use as a normal AST block.
                -;              On the PCB AST queue.
                -;
        <S>     -; System:      The ACB is being processed by system CSP code.
                -;              Not on any queue.
>

CEV$AL_ACTTAB:
  $DSP_TABLE -
    <-
        < 0, ACT_NOP>           -; Nop action routine
        < 2, ACT_BUG>           -; Bugcheck
        < 4, ACT_NYI>           -; Not yet implemented
        <10, ACT_INSQUE>        -; Queue ACB to 'idle' queue, resignal the event
        <12, ACT_REMQUE>        -; Remove ACB from current queue, resignal event
        <14, ACT_REQ_ILL_BT>    -; User requested block transfer on via a CSD
                                -; that is in the wrong state
        <16, ACT_REQ_DEAL>      -; User requested CSD deallocation before AST
                                -; was delivered
        <18, ACT_GET_CDRP>      -; Allocate warm CDRP
        <20, ACT_FORK_WAIT>     -; Put ACB on FORK and WAIT queue
        <22, ACT_BLOCK_XFER>    -; Request ACKMSG block transfer
        <24, ACT_SYN_ERROR>     -; Process synchronous block transfer error
        <26, ACT_QUE_KAST>      -; Request Special Kernel AST
        <28, ACT_QUE_AST>       -; Request Normal  Kernel AST
        <32, ACT_DEALL>         -; Deallocate CSD
        <34, ACT_GIVE_UP>       -; Retry count exceeded
        <36, ACT_NO_AST>        -; No client AST to deliver
    >
```

```
                     0094   331
                     0094   332 CEV$AW_STA_TAB:
                     0094   333 ;
                     0094   334 ;                    I    F    X    K    A    S
                     0094   335 ;           ----------+----+----+----+----+----+----+
                     0094   336 $CEV  EXIT      ?    ?    ?    ?    ?    ?    ; Exit state table processing
                     00A0   337 $CEV  BUG       ?    ?    ?    ?    ?    ?    ; Bug detected
                     00AC   338                                                 ;
                     00AC   339 $CEV  REQ_BT    S12  .14  .14  .14  .14  .18  ; User block-transfer request
                     00B8   340 $CEV  REQ_DEALL S12  .16  .16  .16  .16  .32  ; User's deallocate CSD request
                     00C4   341                                                 ;
                     00C4   342 $CEV  NO_CDRP   ?    ?    ?    ?    ?    F20  ; No CDRP's available
                     00D0   343 $CEV  FORK_DONE ?    S18  ?    ?    ?    ?    ; Back from FORK_WAIT
                     00DC   344 $CEV  GOT_CDRP  ?    ?    ?    ?    ?    X22  ; CDRP was allocated
                     00E8   345 $CEV  BT_DONE   ?    ?    K26  ?    ?    ?    ; Block-transfer done
                     00F4   346 $CEV  BT_SYNERR .24  ?    I10  ?    ?    ?    ; Synchronous transfer error
                     0100   347                                                 ;
                     0100   348 $CEV  CSP_BUSY  ?    ?    F20  ?    ?    ?    ; Remote CSP is busy
                     010C   349 $CEV  NO_CSP    ?    ?    F20  ?    ?    ?    ; No CSP on remote node
                     0118   350 $CEV  GIVE_UP   ?    K34  ?    ?    ?    ?    ; Retry count exceeded
                     0124   351                                                 ;
                     0124   352 $CEV  KAST_DEL  ?    ?    ?    A28  ?    ?    ; Special Kernel AST delivered
                     0130   353 $CEV  AST_DEL   .    ?    ?    ?    I10  ?    ; Normal Kernel AST delivered
                     013C   354 $CEV  NO_AST    .36  ?    ?    I10  S32  ?    ; No user AST to deliver
                     0148   355 $CEV  INV_PID   S12  ?    ?    ?    ?    .32  ; Event is "invalid PID"
                     0154   356
                     0154   357
                     0154   358 ;
                     0154   359 ;  Table to map CSPMSG$K_RSP codes to CEV$_ events
                     0154   360 ;
                     0154   361 CEV$AB_RSP_CEV:
                     0154   362     $RSP_CEV_TAB -
                     0154   363         <-
                     0154   364             <NOP,       BUG>                -; Not supposed to be used
                     0154   365             <BUSY,      CSP_BUSY>           -; Remote CSP is busy, try later
                     0154   366             <NOCSP,     NO_CSP>             -; No CSP process
                     0154   367             <RO,        BT_DONE>            -; Read/only completion
                     0154   368             <RW,        BT_DONE>            -; Read/write completion
                     0154   369             <BADCSD,    BUG>                -; Illegal CSD detected
                     0154   370             <ASYNERR,   BT_DONE>            -; Asynchronous block transfer failure
                     0154   371             <SYNERR,    BT_SYNERR>          -; Synchronous block transfer failure
                     0154   372             <MAX,       BUG>                -; Not supposed to be used
                     0154   373         >
                     015E   374
                     015E   375 ;
                     015E   376 ;  Queue headers
                     015E   377 ;
                     015E   378                     .ALIGN QUAD
                     0160   379
00000000 00000000    0160   380 CSP$Q_ACB_IDLE:   .QUAD  0       ; ACB/CSD's allocated to some process but
                     0168   381                                   ; which are otherwise idle
00000000 00000000    0168   382 CSP$Q_ACB_XFER:   .QUAD  0       ; ACB/CSD's with block transfer in progress
                     0170   383                                   ;
              00     0170   384 CSP$B_RCVCSDCNT: .BYTE  0       ; Number of rcv'd CSD's being processed
                     0171   385                                   ; currently.
              00     0171   386 CSP$B_INITED:    .BYTE  0       ; Zero only if queue's not inited
                     0172   387
```

```
                     0172   388 ;
                     0172   389 ;  Define CSP specific receive CDRP fields and extensions
                     0172   390 ;
00000060             0172   391 CDRP$L_CSP_CSD =  0+CDRP$K_CM_LENGTH    ; Pointer to allocated CSD
00000064             0172   392 CDRP$L_CSP_SP1 =  4+CDRP$L_CSP_CSD      ; Spare
                     0172   393
                     0172   394 $VIELD  CDRP,0,-                       ; Define CDRP$B_CLTSTS flags
                     0172   395 <-
                     0172   396         <CSP_ERROR,,M>,-                ; ACKMSG error experienced
                     0172   397         <CSP_QUEUED,,M>,-               ; CSD is queued to CSP process
                     0172   398         <CSP_FLWCTL,,M>,-               ; CSD accounted against flow control
                     0172   399 >
                     0172   400
                     0172   401
                     0172   402         .PSECT  $$$200,EXE              ; Go to code .PSECT
                     0172   403
```

```
                     0172   405 .SBTTL  'CSP$INIT       - Init CSP data structures upon load'
                     0172   406 ;++
                     0172   407 ;
                     0172   408 ;  This code is called once when the CLUSTRLOA is loaded.  It init's the
                     0172   409 ;  queue headers.
                     0172   410 ;
                     0172   411 ;  INPUTS:      NONE
                     0172   412 ;
                     0172   413 ;  OUTPUTS:     R0      SS$_NORMAL
                     0172   414 ;
                     0172   415 ;--
                     0172   416 CSP$INIT::                                  ; Init data structures
      25 FC AF    E8 0172   417         BLBS    CSP$B_INITED,100$           ; If LBS, we've been here
                     0176   418
                     0176   419         ASSUME  CSP$Q_ACB_XFER  EQ  8+CSP$Q_ACB_IDLE
                     0176   420
   50    E7 AF    9E 0176   421         MOVAB   CSP$Q_ACB_IDLE,R0           ; Get queue header address
      80    60    9E 017A   422         MOVAB   (R0),(R0)+                  ; Setup ACB_IDLE queue header
   80    FC A0    9E 017D   423         MOVAB   -4(R0),(R0)+                ;
      80    60    9E 0181   424         MOVAB   (R0),(R0)+                  ; Setup ACB_XFER queue header
   80    FC A0    9E 0184   425         MOVAB   -4(R0),(R0)+                ;
                     0188   426                                             ;
 00000088 8F    C1 0188   427         ADDL3   #CLUB$L_CSPFL,-             ; Get queue header address
50   00000000'GF          018E   428                 G^CLU$GL_CLUB,R0        ;
      60    60    9E 0194   429         MOVAB   (R0),(R0)                   ; Setup forward link
   04 A0    60    9E 0197   430         MOVAB   (R0),4(R0)                  ; Setup backward link
                     019B   431                                             ;
   D2 AF    01    90 019B   432 100$:   MOVB    #1,CSP$B_INITED             ; Say "initialized"
      50    01    D0 019F   433         MOVL    #SS$_NORMAL,R0              ; Always successful
                  05 01A2   434         RSB                                 ; Done
                     01A3   435
```

```
                        01A3    437 .SBTTL  'CLEAN_UP       - ACKMSG Rcv cleanup routine'
                        01A3    438 ;++
                        01A3    439 ;
                        01A3    440 ;  This routine is called by ACKMSG when a fatal virtual circuit error is
                        01A3    441 ;  encountered.  ACKMSG is going to drop this thread on the floor and will
                        01A3    442 ;  deallocate the CLUBTX structure.  It is up to us to eventually deallocate
                        01A3    443 ;  the CDRP and the CSD.
                        01A3    444 ;
                        01A3    445 ;  INPUTS:       R5      CDRP Pointer
                        01A3    446 ;                 R4      N/A
                        01A3    447 ;                 R3      CSB (or zero)
                        01A3    448 ;                 R2      Pointer to message stored in CLUBTX
                        01A3    449 ;                 R1      Pointer to extension space at end of CLUBTX (0 if none)
                        01A3    450 ;                 R0      Scratch
                        01A3    451 ;
                        01A3    452 ;  OUTPUTS:      ??
                        01A3    453 ;
                        01A3    454 ;--
                        01A3    455                   .ENABL  LSB
                        01A3    456 CLEAN_UP:                                                   ; Cleanup upon error
    4B A5   01    88    01A3    457                   BISB    #CDRP$M_CSP_ERROR,CDRP$B_CLTSTS(R5) ; Remember error
                        01A7    458 CLEAN_UP1:                                                  ; Internal cleanup
20 4B A5    01    E0    01A7    459                   BBS     #CDRP$V_CSP_QUEUED,CDRP$B_CLTSTS(R5),100$ ; If BS, CSD is
                        01AC    460                                                             ; queued to CSP
03 4B A5    02    E5    01AC    461                   BBCC    #CDRP$V_CSP_FLWCTL,CDRP$B_CLTSTS(R5),50$ ; If BS, accounted
                        01B1    462                                                             ; against flow control
      BC AF     97    01B1    463                   DECB    CSP$B_RCVCSDCNT                   ; Return flow credit
   50   60 A5    D0    01B4    464 50$:              MOVL    CDRP$L_CSP_CSD(R5),R0             ; Get CSD
        09        13    01B8    465                   BEQL    70$                               ; If EQL, none
      60 A5       D4    01BA    466                   CLRL    CDRP$L_CSP_CSD(R5)                ; Clear ptr
00000000'GF       16    01BD    467                   JSB     G^EXE$DEANONPAGED                 ; Deallocate CSD
   50   55        D0    01C3    468 70$:              MOVL    R5,R0                             ; Get CDRP
00000000'GF       16    01C6    469                   JSB     G^EXE$DEANONPAGED                 ; Deallocate CDRP
        05        05    01CC    470 100$:             RSB                                       ; Done
                        01CD    471
                        01CD    472                   .DSABL  LSB
                        01CD    473
```

```
                                01CD    475 .SBTTL  'CSP$DISPATCH   - Dispatch on received ACKMSG message'
                                01CD    476 ;++
                                01CD    477 ;
                                01CD    478 ;  INPUTS:      R5      Unitialized CDRP
                                01CD    479 ;               R4      PDT address
                                01CD    480 ;               R3      CSB address
                                01CD    481 ;               R2      Message address
                                01CD    482 ;               R1-R0   Scratch
                                01CD    483 ;
                                01CD    484 ;  OUTPUTS:     R5-R0   Garbage
                                01CD    485 ;
                                01CD    486 ;--
                                01CD    487                 .ENABL  LSB
                                01CD    488 CSP$DISPATCH::                                  ; CSP ACMKSG dispatcher
                                01CD    489                 ;
                                01CD    490                 ;
                                01CD    491                 ;   Call CNX$PARTNER_INIT_CSB to allocate new BTX (R2) and to init CDRP
                                01CD    492                 ;
                                01CD    493                 ;
                      51    D4  01CD    494                 CLRL    R1                          ; No BTX extension space needed
                54    D1 AF   9E  01CF    495                 MOVAB   CLEAN_UP,R4                 ; Address of cleanup routine
                        FE2A'   30  01D3    496                 BSBW    CNX$PARTNER_INIT_CSB        ; Prepare for block transfer
                                01D6    497                                                     ;  - may return to our caller
                                01D6    498                                                     ;  - may never return if
                                01D6    499                                                     ;    connection breaks
                      4B A5     94  01D6    500                 CLRB    CDRP$B_CLTSTS(R5)           ; Init client (us) status
                      60 A5     D4  01D9    501                 CLRL    CDRP$L_CSP_CSD(R5)          ; Init CSD pointer
                      64 A5     D4  01DC    502                 CLRL    CDRP$L_CSP_SP1(R5)          ; Init spare longword
                   51   02     D0  01DF    503                 MOVL    #CSP$_ABORT,R1              ; Say "no CSP process"
             50 00000000'GF     D0  01E2    504                 MOVL    G^CLU$GL_CLUB,R0            ; Get CLUB
                      10     13  01E9    505                 BEQL    20$                         ; If EQL, none
                  0090 C0     D5  01EB    506                 TSTL    CLUB$L_CSPIPID(R0)          ; CSP there ?
                      0A     13  01EF    507                 BEQL    20$                         ; If EQL, no
                FF7A CF       08  91  01F1    508                 CMPB    #CSP$K_MAX_FLWCTL,CSP$B_RCVCSDCNT   ; Within limit?
                      09     1A  01F6    509                 BGTRU   30$                         ; If GTRU yes, okay to continue
                   51   06     D0  01F8    510 10$:            MOVL    #CSP$_REJECT,R1             ; "reject due to flow control"
                      010A    30  01FB    511 20$:            BSBW    CSP_COMMAND                 ; Issue command
                      008C    31  01FE    512                 BRW     100$                        ; Done
                                0201    513 30$:            ;
                                0201    514                 ;
                                0201    515                 ;   Flow control allows us to continue.  Allocate a CSD to receive the
                                0201    516                 ;   remote request.
                                0201    517                 ;
                                0201    518                 ;
                3C A5     1C A2     D0  0201    519                 MOVL    CSPMSG$L_CSD_SIZE(R2),CDRP$L_XCT_LEN(R5); Save CSD size
             51 3C A5     0C     C1  0206    520                 ADDL3   #12,CDRP$L_XCT_LEN(R5),R1   ; Get total CSD size
                00000000'GF     16  020B    521                 JSB     G^EXE$ALONONPAGED           ; Allocate CSD
                      E4 50     E9  0211    522                 BLBC    R0,10$                      ; If LBC no, treat as
                                0214    523                                                     ; flow control problem
                   FF58 CF      96  0214    524                 INCB    CSP$B_RCVCSDCNT             ; Consume flow control
                4B A5     04     88  0218    525                 BISB    #CDRP$M_CSP_FLWCTL,CDRP$B_CLTSTS(R5)    ; And mark the fact
                                021C    526                 ;
                                021C    527                 ;
                                021C    528                 ;   Setup the CDRP for the block transfer, and read the remote command
                                021C    529                 ;   into the allocated buffer.
                                021C    530                 ;
                                021C    531                 ;   The call to CNX$BLOCK_READ returns to our caller immediately, and
```

```
                                021C   532          ;    returns in-line only after the transfer completes.  If an error is
                                021C   533          ;    encountered and our error routine (CLEAN_UP) is called, then there
                                021C   534          ;    is no return in-line.
                                021C   535          ;
                                021C   536          ;
          60 A5    52    D0     021C   537          MOVL    R2,CDRP$L_CSP_CSD(R5)                   ; Setup pointer
          08 A2    51    3C     0220   538          MOVZWL  R1,8(R2)                                ; Setup size
             62    55    D0     0224   539          MOVL    R5,(R2)                                 ; Setup CDRP pointer
             52    0C    C0     0227   540          ADDL    #12,R2                                  ; Go to CSD area
     51   52    15    09    EF     022A   541          EXTZV   #VA$V_VPN,#VA$S_VPN,R2,R1               ; Get page number
      50   00000000'GF    D0     022F   542          MOVL    G^MMG$GL_SPTBASE,R0                     ; Get base of SPT
        40 A5    6041    DE     0236   543          MOVAL   (R0)[R1],CDRP$L_CNXSVAPTE(R5)           ; Setup SVAPTE
44 A5   52    FE00 8F    AB     023B   544          BICW3   #^C<VA$M_BYTE>,R2,CDRP$W_CNXBOFF(R5)      ; Setup BOFF
      46 A5    3C A5    D0     0242   545          MOVL    CDRP$L_XCT_LEN(R5),CDRP$L_CNXBCNT(R5)   ; Setup BCNT
            4A A5    94     0247   546          CLRB    CDRP$B_CNXRMOD(R5)                      ; Setup for kernel mode
            38 A5    D4     024A   547          CLRL    CDRP$L_RBOFF(R5)                        ; Start at begining of
            30 A5    D4     024D   548          CLRL    CDRP$L_LBOFF(R5)                        ;  buffer on both sides
              FDAD'  30     0250   549          BSBW    CNX$BLOCK_READ                          ; Read remote request
                                0253   550          ;
                                0253   551          ;
                                0253   552          ;    We only get here if the READ completed successfully.  Pickup the
                                0253   553          ;    CSD, queue it, and wake the CSP process to come and get it.
                                0253   554          ;
                                0253   555          ;    If the CSP is no longer there (SCH$WAKE fails), empty the CSD queue
                                0253   556          ;    and send an approriate response.
                                0253   557          ;
                                0253   558          ;
   52   60 A5    0C    C1     0253   559          ADDL3   #12,CDRP$L_CSP_CSD(R5),R2               ; Get the CSD
        4B A5    02    88     0258   560          BISB    #CDRP$M_CSP_QUEUED,CDRP$B_CLTSTS(R5)    ; Say "queued to CSP"
                                025C   561          ;
                                025C   562 INSQUE_CLUB:                                          ; Queue CSD to CLUB
                                025C   563          ;
                                025C   564          ;
                                025C   565          ;    Inputs:     R0      Scratch
                                025C   566          ;                R1      Scratch
                                025C   567          ;                R2      CSD pointer
                                025C   568          ;                R3      Scratch
                                025C   569          ;                R4      Scratch
                                025C   570          ;                R5      CDRP pointer, if any
                                025C   571          ;
                                025C   572          ;
50   00000000'GF    D0     025C   573          MOVL    G^CLU$GL_CLUB,R0                        ; Get CLUB
   008C D0    62    0E     0263   574          INSQUE  (R2),@CLUB$L_CSPBL(R0)                  ; Queue the CSD
   51   0090 C0    D0     0268   575          MOVL    CLUB$L_CSPIPID(R0),R1                   ; Get CSP's IPID
             09    13     026D   576          BEQL    80$                                     ; If EQL, no CSP
    00000000'GF    16     026F   577          JSB     G^SCH$WAKE                              ; Wake CSP
          15 50    E8     0275   578          BLBS    R0,100$                                 ; If LBS, okay
54   00000000'GF    D0     0278   579 80$:     MOVL    G^CLU$GL_CLUB,R4                        ; Get the CLUB
   52   0088 D4    0F     027F   580 90$:     REMQUE  @CLUB$L_CSPFL(R4),R2                    ; Get the CSD
             07    1D     0284   581          BVS     100$                                    ; If VS, none left
          51   02    D0     0286   582          MOVL    #CSP$_ABORT,R1                          ; Setup function code
             03    10     0289   583          BSBB    EXE$CSP_COMMAND                         ; Process CSD
             F2    11     028B   584          BRB     90$                                     ; Loop
             05    028D   585 100$:    RSB                                                    ; Done
                   028E   586                  .DSABL  LSB
                   028E   587
```

```
                     028E    589 .SBTTL  'EXE$CSP_COMMAND  Receive commnad from CSP process'
                     028E    590 ;++
                     028E    591 ;
                     028E    592 ;  The CSP process calls this routine when it is done processing a CSD.  The
                     028E    593 ;  action is to conditionally send the CSD back to the requestor (if it contains
                     028E    594 ;  new data) and to terminate the block transfer sequence with a response
                     028E    595 ;  message.
                     028E    596 ;
                     028E    597 ;  This routine is also used to process the CSP$_LOCAL command.  This command
                     028E    598 ;  is used to pass locally generated requests to the CSP process.
                     028E    599 ;
                     028E    600 ;  INPUTS:      R4      client code     (CSP$_LOCAL only)
                     028E    601 ;               R3      0              (CSP$_LOCAL only)
                     028E    602 ;                       Will someday be used for message build call back
                     028E    603 ;               R2      Address of CSD
                     028E    604 ;               R1      Function code:
                     028E    605 ;
                     028E    606 ;                           CSP$_ABORT  - Abort the request
                     028E    607 ;                           CSP$_BADCSD - Illegal CSD structure detected
                     028E    608 ;                           CSP$_DONE   - Terminate the exchange
                     028E    609 ;                           CSP$_REJECT - Reject request due to flow control
                     028E    610 ;                           CSP$_REPLY  - Send CSD back to requestor
                     028E    611 ;                           CSP$_LOCAL  - Send local CSD to CSP
                     028E    612 ;
                     028E    613 ;               R0      Scratch
                     028E    614 ;
                     028E    615 ;  OUTPUTS:     R2-R0   Garbage
                     028E    616 ;
                     028E    617 ;--
                     028E    618 EXE$CSP_COMMAND::                                   ; Command from CSP
                  38 BB      028E    619         PUSHR   #^M<R3,R4,R5>               ; Save regs
                     0290    620         DSBINT  #IPL$_SYNCH                         ; Go to proper IPL
                     0296    621                                                     ;
                  06 10      0296    622         BSBB    50$                         ; Process the command
                     0298    623                                                     ;
                     0298    624         ENBINT                                      ; Restore IPL
                  38 BA      029B    625         POPR    #^M<R3,R4,R5>               ; Restore regs
                  05         029D    626         RSB                                 ; Done
                     029E    627
               07 51 D1      029E    628 50$:    CMPL    R1,#CSP$_LOCAL              ; "Local" request ?
                  4D 12      02A1    629         BNEQ    CSP_COMMAND_1               ; If NEQ, no
                     02A3    630                 ;
                     02A3    631                 ;
                     02A3    632                 ;    This is a "local" request
                     02A3    633                 ;
                     02A3    634                 ;
            FEC8 CF  08 91   02A3    635         CMPB    #CSP$K_MAX_FLWCTL,CSP$B_RCVCSDCNT  ; Within limit?
                  07 1A      02A8    636         BGTRU   70$                         ; If GTRU, okay
            50  0294 8F 3C   02AA    637 60$:    MOVZWL  #SS$_REJECT,R0              ; Tell caller we failed
                  3E 11      02AF    638         BRB     100$                        ; Take common exit
            51  005E 8F 3C   02B1    639 70$:    MOVZWL  #12+CSD$K_LENGTH,R1         ; Setup block size
              00000000'GF    02B6    640         JSB     G^EXE$ALONONPAGED           ; Allocate the block
                     EB 50   02BC    641         BLBC    R0,60$                      ; If LBC, failed
                     02BF    642                 ;
                  3F BB      02BF    643         PUSHR   #^M<R0,R1,R2,R3,R4,R5>      ; Save regs
62 51 00 6E       00 2C      02C1    644         MOVC5   #0,(SP),#0,R1,(R2)          ; Zero the block
                  3F BA      02C7    645         POPR    #^M<R0,R1,R2,R3,R4,R5>      ; Restore regs
```

```
                        02C9    646
                        02C9    647          MOVZWL  R1,8(R2)                        ; Setup size, zero type
                        02CD    648          ADDL    #12,R2                          ; Goto CSD area
                        02D0    649          SUBL    #12,R1                          ; Reduce size
                        02D3    650          MOVW    R1,8(R2)                        ; Setup size
                        02D7    651          MOVB    #DYN$C_CLU,CSD$B_TYPE(R2)       ; Setup type
                        02DC    652          MOVB    #DYN$C_CSD,CSD$B_SUBTYPE(R2)    ; Setup subtype
                        02E1    653          MOVW    R4,CSD$W_CODE(R2)               ; Enter client code
                        02E5    654          INCB    CSP$B_RCVCSDCNT                 ; Consume flow control
                        02E9    655          BSBW    INSQUE_CLUB                     ; Queue the CSD
                        02EC    656          ;
                        02EC    657          ;
                        02EC    658          ;   *** NOTE ***
                        02EC    659          ;
                        02EC    660          ;   For a variety of reasons (CSP not there yet, CSP was there when
                        02EC    661          ;   CSD was queued but exitted shortly thereafter), a return with
                        02EC    662          ;   the low bit set does not mean that the request actually made
                        02EC    663          ;   it.  A return with the low bit clear does mean that it didn't.
                        02EC    664          ;
                        02EC    665          ;   A more sophisticated mechanism for status reporting will need
                        02EC    666          ;   to be invented if this is not adequate for future users of
                        02EC    667          ;   this interface (currenly only the Quorum disk thread uses this).
                        02EC    668          ;
                        02EC    669          ;
     50    01    D0     02EC    670          MOVL    #1,R0                           ; Assume success (error at
                        02EF    671                                                  ; this point is untrustworthy)
                  05    02EF    672  100$:    RSB                                     ; Return status to caller
                        02F0    673
                        02F0    674
                        02F0    675  CSP_COMMAND_1:                                 ; Process CSP command
                        02F0    676          ;
                        02F0    677          ;
                        02F0    678          ;   If the CDRP pointer is zero, then this is a "local" CSD being
                        02F0    679          ;   returned -- simply restore the flow control taken and deallocate
                        02F0    680          ;   the CSD.  Otherwise,
                        02F0    681          ;
                        02F0    682          ;
   55    F4  A2    D0     02F0    683          MOVL    -12(R2),R5                      ; Get CDRP
               0E    12     02F4    684          BNEQ    5$                              ; If NEQ, not local CSD
         FE76  CF    97     02F6    685          DECB    CSP$B_RCVCSDCNT                 ; Restore flow control
   50    F4  A2    9E     02FA    686          MOVAB   -12(R2),R0                      ; Get block address
 00000000'GF       17     02FE    687          JMP     G^EXE$DEANONPAGED               ; Deallocate the block
               02  8A     0304    688  5$:     BICB    #CDRP$M_CSP_QUEUED,-            ; CSP is done with CSD
         4B  A5            0306    689                   CDRP$B_CLTSTS(R5)
                           0308    690  CSP_COMMAND:                                  ; Process CSP command
                           0308    691                                                ;
38  4B  A5    00    E0     0308    692          BBS     #CDRP$V_CSP_ERROR,CDRP$B_CLTSTS(R5),900$ ; If BS, ACKMSG error
                           030D    693                                                ; occurred
                           030D    694          DISPATCH  R1,-
                           030D    695          <-
                           030D    696              <CSP$_DONE,   100$>,-           ; Terminate the exchange
                           030D    697              <CSP$_BADCSD, 300$>,-           ; Illegal CSD structure
                           030D    698              <CSP$_ABORT,  310$>,-           ; CSP is not there or is going
                           030D    699              <CSP$_REJECT, 320$>,-           ; Reject due to no flow control
                           030D    700              <CSP$_REPLY,  800$>,-           ; Send CSD back to requestor
                           030D    701          >
                           031B    702
```

```
                     031B   703          BUG_CHECK  INCONSTATE,FATAL                ; Unknown command
                     031F   704
                     031F   705 100$:     ;
                     031F   706           ;
                     031F   707           ;    Send CSD back to requestor before finishing up the block transfer
                     031F   708           ;
                     031F   709           ;
        FCDE'   30   031F   710          BSBW    CNX$BLOCK_WRITE                  ; Send CSD back to requestor
     51   05   D0   0322   711          MOVL    #CSPMSG$K_RSP_RW,R1              ; Setup response code
          12   11   0325   712          BRB     810$                             ; Finish up block transfer
                     0327   713
                     0327   714           ;
                     0327   715           ;
                     0327   716           ;    Miscellaneous failures
                     0327   717           ;
                     0327   718           ;
     51   06   D0   0327   719 300$:     MOVL    #CSPMSG$K_RSP_BADCSD,R1          ; Inidicate 'bad csd'
          0D   11   032A   720          BRB     810$                             ; Finish up block transfer
     51   03   D0   032C   721 310$:     MOVL    #CSPMSG$K_RSP_NOCSP,R1           ; Indicate 'no CSP process'
          08   11   032F   722          BRB     810$                             ; Finish up block transfer
     51   02   D0   0331   723 320$:     MOVL    #CSPMSG$K_RSP_BUSY,R1            ; Indicate 'no flow credits'
          03   11   0334   724          BRB     810$                             ; Finish up block transfer
                     0336   725
                     0336   726 800$:     ;
                     0336   727           ;
                     0336   728           ;    Finish up the block transfer and deallocate the CDRP and CSD
                     0336   729           ;    Store the response code in low byte of CDRP$L_VAL2.
                     0336   730           ;
     51   04   D0   0336   731          MOVL    #CSPMSG$K_RSP_RO,R1              ; Setup response code
30   A5   51   90   0339   732 810$:     MOVB    R1,CDRP$L_VAL2(R5)               ; Enter response code
4C   A5   49'AF   9E   033D   733          MOVAB   B^RSP_MSGBLD,CDRP$L_MSGBLD(R5)   ; Setup message build routine
          FCBB'   30   0342   734          BSBW    CNX$PARTNER_RESPOND              ; Finish up block transfer
          FE5F    30   0345   735 900$:     BSBW    CLEAN_UP1                        ; Cleanup CDRP, CSD, etc
                  05   0348   736          RSB                                      ; Done
                     0349   737
                     0349   738 RSP_MSGBLD:
                     0349   739           ;
                     0349   740           ;    ACKMSG calls us here to build the response message.
                     0349   741           ;
                     0349   742           ;    INPUTS:     R5      CDRP ptr
                     0349   743           ;                R4      PDT ptr
                     0349   744           ;                R3      CSB ptr
                     0349   745           ;                R2      Message pointer
                     0349   746           ;                R0      Scratch
                     0349   747           ;
                     0349   748           ;
                     0349   749                                                    ;
18   A2   30   A5   90   0349   750          MOVB    CDRP$L_VAL2+0(R5),CSPMSG$B_RSP(R2)   ; Copy CSP response
08   A2   86   8F   90   034E   751          MOVB    #<CLSMSG$K_FAC_CSP ! CLSMSG$M_RESPMSG>, -   ; Copy code/flag
                     0353   752                  CLSMSG$B_FACILITY(R2)
          09   A2   94   0353   753          CLRB    CLSMSG$B_FUNC(R2)                ; Copy our fct
                  05   0356   754          RSB                                      ; Done
                     0357   755
```

```
                        .SBTTL  'EXE$CSP_BRDCST - Send CSP request to all nodes'
                        ;++
                        ;
                        ;  Send specified message to all other nodes in the cluster.  A list is made of
                        ;  all nodes currently in the cluster, and the message is sent to the CSP in
                        ;  each.  A new list is then made and compared with the first; if any new nodes
                        ;  have appeared, the message is sent to them.  This repeats until the no new
                        ;  nodes appear.  Note that the local node is excluded from the list of
                        ;  recipients.
                        ;
                        ;
                        ;  Allocation and Deallocation of CSD's
                        ;  ------------------------------------
                        ;
                        ;  EXE$ALLOC_CSD   should be used to allocate all CSD's.
                        ;  EXE$DEALLOC_CSD should be used to deallocate all CSD's.
                        ;
                        ;  Because some fields in the CSD need reinitializing, and since the call to
                        ;  EXE$DEALLOC_CSD is merely a request (the actual deallocation can only happen
                        ;  when the CSD "runs down"), CSD's should not be recycled by the clients, but
                        ;  rather a fresh one should be allocated for each use.
                        ;
                        ;  The template CSD is allocated by the caller and this routine allocates the
                        ;  rest.  However, the AST routine is responsible for deallocating each CSD;
                        ;  this is true of every CSD the AST routine is called with, including the
                        ;  template CSD.  If there is no AST routine specified, then EXE$CSP_BRDCST will
                        ;  cause the CSD's used in the node dialogues to be automatically deallocated.
                        ;  Note that the AST routine need not deallocate a CSD immediately -- it may
                        ;  queue for later deallocation at normal process level.
                        ;
                        ;  The caller is always responsible for deallocating the template CSD as listed
                        ;  in the table below.  Basically, if the call to this routine returns an error,
                        ;  or if no AST is specified, then the caller should deallocate the CSD upon
                        ;  return.  Otherwise, the AST routine should cause the CSD (in this case
                        ;  CSD$L_CSID = -1) to be deallocated.
                        ;
                        ;
                        ;  The CSD$L_USER_AST field
                        ;  -----------------------
                        ;
                        ;  If this field is zero, then no AST's will be delivered and control is not
                        ;  returned to the caller until the completion of the dialogue with the final
                        ;  node.
                        ;
                        ;  If this field is non-zero, then control is returned to the user as soon as
                        ;  possible.  An AST will be delivered after the completion of a dialogue with
                        ;  each node.  The CSD address is the AST parameter.  The AST routine should
                        ;  check the CSD$L_CSID field to determine the remote node, and CSD$Q_INT_IOSB
                        ;  to determine the status.  Also, it may read the response data described by
                        ;  CSD$L_RECVLEN and CSD$L_RECVOFF.
                        ;
                        ;  If EXE$CSP_BRDCST returns with the low bit set in R0, then an AST will be
                        ;  delivered using the template CSD as a parameter (i.e, CSD$L_CSID=-1) after
                        ;  completion of the dialogue with the final node.  This allows the caller to
                        ;  know when the all of the EXE$CSP_BRDCST operations are done.
                        ;
```

```
0357    814 ;  If EXE$CSP_BRDCST returns with the low bit clear in R0, then no further AST
0357    815 ;  will be queued to the process (those already in the queue will be delivered
0357    816 ;  when process state allows).  This means that the AST routine will not be
0357    817 ;  called with the template CSD.
0357    818 ;
0357    819 ;
0357    820 ;  Danger of Disabling AST's
0357    821 ;  ------------------------
0357    822 ;
0357    823 ;  Since the allocation of CSD's is charged against the user's BYTCNT quota,
0357    824 ;  and if the caller has specified an AST routine, then calling EXE$CSP_BRDCST
0357    825 ;  could hang the process.  This is because the quota is only returned when a
0357    826 ;  CSD is deallocated, and that does not happen until the AST causes to happen.
0357    827 ;  This also implies that the CSD should be deallocated as soon as possible
0357    828 ;  after the AST is delivered.
0357    829 ;
0357    830 ;  AST's may be disabled if no AST routine is specified since in that case
0357    831 ;  an AST does not have to be delivered before the quota is returned since the
0357    832 ;  CSD is deallocated in the 'Special Kernel' AST routine that is delivered
0357    833 ;  when the block transfer completes or fails.  Note that 'Special Kernel' AST's
0357    834 ;  are not disabled by the $SETAST service.
0357    835 ;
0357    836 ;
0357    837 ;  Waiting for Pool or Process Quota
0357    838 ;  --------------------------------
0357    839 ;
0357    840 ;  When system resources or process quotas are not available, EXE$CSP_BRDCST
0357    841 ;  will optionally wait, depending on the setting of PCB$V_SSRWAIT, in the
0357    842 ;  current mode (kernel) at IPL 0.  This will allow the process to be deleted
0357    843 ;  (cleanup any allocated pool is eventually done when the timer ticks or some
0357    844 ;  block transfer completes), but will not allow the user to "^Y, STOP" the
0357    845 ;  current running image.  The later problem should be solved someday, but it
0357    846 ;  it is non-trivial since our caller is not the "user" but is some internal
0357    847 ;  system service code which may have resources to clean up.
0357    848 ;
0357    849 ;       NOTE:   Caller's of this routine are therefore cautioned from making
0357    850 ;               this eventual solution overly difficult by calling
0357    851 ;               EXE$CSP_BRDCST from awkward places.
0357    852 ;
0357    853 ;
0357    854 ;
0357    855 ;  In summary
0357    856 ;  ----------
0357    857 ;
0357    858 ;  R0's     AST        When to EXE$DEALLOC_CSD          When EXE$CSP_BRDCST
0357    859 ;  low bit  specified the template CSD                 returns to caller
0357    860 ;  -------  --------- ------------------------         -------------------
0357    861 ;    LBC      no      Upon return - no further         When the error is
0357    862 ;                     AST's are delivered.             encountered.
0357    863 ;    LBC      yes     Upon return - no further         When the error is
0357    864 ;                     AST's are delivered.             encountered.
0357    865 ;    LBS      no      Upon return                      When all dialogues
0357    866 ;                                                      have completed.
0357    867 ;    LBS      yes     By the AST routine or by         As soon as possible.
0357    868 ;                     some action it schedules.
0357    869 ;
0357    870 ;
```

```
                                0357   871 ;
                                0357   872 ;
                                0357   873 ;  CALLING SEQUENCE:    JSB     EXE$CSP$BRDCST  at IPL 0
                                0357   874 ;
                                0357   875 ;
                                0357   876 ;  INPUTS:       R2      Address of template CSD which is completely filled in
                                0357   877 ;                        (including user data) with the exception CSD$L_CSID.
                                0357   878 ;
                                0357   879 ;  OUTPUTS:      R0      Status
                                0357   880 ;
                                0357   881 ;                  All other registers are preserved.
                                0357   882 ;
                                0357   883 ;--
                                0357   884 EXE$CSP_BRDCST::
                    03FE 8F     BB  0357   885          PUSHR   #^M<R1,R2,R3,R4,R5,R6,R7,R8,R9> ; Save volatile reg's
                                035B   886                                                     ;
                      0216      30  035B   887          BSBW    COMMON_SETUP                   ; Check IPL, get ACB, etc
                    7D 50       E9  035E   888          BLBC    R0,100$                        ; If LBC, error
                56     52       D0  0361   889          MOVL    R2,R6                          ; Save ptr to the template CSD
                59     54       D0  0364   890          MOVL    R4,R9                          ; Save ACB pointer
                  28 A9          B6  0367   891          INCW    ACB$W_WAIT_CNT(R9)             ; Bias the wait count while
                                036A   892                                                     ; this routine is using the ACB
            50    028C 8F       3C  036A   893          MOVZWL  #SS$_NOSUCHNODE,R0             ; set up other escape code
                                036F   894                                                     ;
              0E A6    01       CE  036F   895          MNEGL   #1,CSD$L_CSID(R6)              ; Mark CSD as "template"
        2A A9   00000000'GF     B0  0373   896          MOVW    G^CLU$GW_MAXINDEX,ACB$W_LAST_INX(R9) ; Init final CSB index
                                037B   897                                                     ;
                                037B   898 10$:     ;
                                037B   899          ;
                                037B   900          ;    Get the next CSB.  If there is one, allocate a CSD and copy the
                                037B   901          ;    the template to it.
                                037B   902          ;
                                037B   903          ;
                      66        10  037B   904          BSBB    GET_NEXT_CSB                   ; Get next CSB, if any
                  48 50         E9  037D   905          BLBC    R0,70$                         ; If LBC, we're done
               51    08 A6      3C  0380   906          MOVZWL  CSD$W_SIZE(R6),R1              ; Get the allocation size
            00000435'GF         16  0384   907          JSB     G^EXE$ALLOC_CSD                ; Get a new CSD for this node
                  3B 50         E9  038A   908          BLBC    R0,70$                         ; Error if LBC (no recovery)
                                038D   909                                                     ;
                      52        DD  038D   910          PUSHL   R2                             ; Save its address
            62    66    51      28  038F   911          MOVC3   R1,(R6),(R2)                   ; Fill it in from the template
                      52  8ED0    0393   912          POPL    R2                             ; Retrieve the CSD
                                0396   913                                                     ;
                                0396   914          ;
                                0396   915          ;
                                0396   916          ;    Make the CSP call to tranfer the CSD.
                                0396   917          ;
                                0396   918          ;
                54    CC A2     9E  0396   919          MOVAB   -ACB$K_CSPLNG(R2),R4           ; Get ACB
                2C A4   59      D0  039A   920          MOVL    R9,ACB$L_PARENT(R4)            ; Remember parent
                  28 A9         B6  039E   921          INCW    ACB$W_WAIT_CNT(R9)             ; Account for this broadcast
                31 A4   18      88  03A1   922          BISB    #ACB$M_STS_BCST!-              ; Mark it as part of broadcast
                                03A5   923                  ACB$M_STS_PCNT,ACB$B_STS(R4)   ; and part of broadcast count
                0E A2   58      D0  03A5   924          MOVL    R8,CSD$L_CSID(R2)              ; Fill in CSID
              0000051E'GF       16  03A9   925          JSB     G^EXE$CSP_CALL                 ; Send it to its fate
                    C9 50       E8  03AF   926          BLBS    R0,10$                         ; Loop if ok
            06 31 A4    04      E5  03B2   927          BBCC    #ACB$V_STS_PCNT,ACB$B_STS(R4),60$; If BC, no longer part of count
```

```
                                03B7   928          CLRL    ACB$L_PARENT(R4)                ; Erase pointer
                                03BA   929          DECW    ACB$W_WAIT_CNT(R9)              ; Account for this broadcast
                                03BD   930 60$:      MOVL    R2,R0                           ; Set for deallocation
                                03C0   931          JSB     G^EXE$DEALLOC_CSD               ; Deallocate
                                03C6   932          BRB     10$                             ; Loop
                                03C8   933 70$:      ;
                                03C8   934          ;
                                03C8   935          ;    We're done.
                                03C8   936          ;
                                03C8   937          ;
                                03C8   938          MOVL    #SS$_NORMAL,R0                  ; Indicate success
                                03CB   939          ;                                       ;
                                03CB   940 80$:      BLBC    R0,100$                         ; If LBC, return immediately
                                03CE   941          DECW    ACB$W_WAIT_CNT(R9)              ; Take back this routine's
                                03D1   942          ;                                       ; reference
                                03D1   943          BNEQ    90$                             ; If NEQ, may need to wait
                                03D3   944          BBCC    #ACB$V_STS_WAIT,ACB$B_STS(R9),90$ ; Else our waiting is done
                                03D8   945 90$:      MOVL    R6,R2                           ; Setup original CSD address
                                03DB   946          BSBW    WAIT                            ; Wait if necessary
                                03DE   947          ;                                       ;
                                03DE   948 100$:     POPR    #^M<R1,R2,R3,R4,R5,R6,R7,R8,R9> ; Restore registers
                                03E2   949          RSB                                     ; Done
                                03E3   950
                                03E3   951
                                03E3   952 GET_NEXT_CSB:
                                03E3   953          DSBINT  #IPL$_SCS                       ; Lock cluster database
                                03E9   954          ;                                       ;
                                03E9   955          CLRL    R0                              ; Assume no new CSB's
                                03EB   956          MOVZWL  ACB$W_LAST_INX(R9),R1           ; Get next index to use
                                03EF   957          BEQL    60$                             ; If EQL, done
00000000'GF                     03F1   958          MOVL    G^CLU$GL_CLUSVEC,R7             ; Address the cluster vector
                                03F8   959          BEQL    60$                             ; If EQL, none
00000000'GF                     03FA   960          MOVL    G^CLU$GL_CLUB,R4                ; Get Cluster Block
                                0401   961          BEQL    60$                             ; If EQL, not in cluster (?)
00000000'GF                     0403   962          MOVZWL  G^CLU$GW_MAXINDEX,R3            ; Get vector length counter
                                040A   963          BEQL    60$                             ; If EQL, none
                                040C   964          CMPW    R3,R1                           ; Compare against last index
                                040F   965          BGEQU   30$                             ; If LSSU, it shrunk
                                0411   966          MOVL    R3,R1                           ; Update current index
                6741            0414   967 30$:     MOVL    (R7)[R1],R2                     ; Get CSB
                                0418   968          BGEQ    50$                             ; If GEQ, this slot is empty
                4C              041A   969          MOVL    CSB$L_CSID(R2),R8               ; Get the CSID
                10              041E   970          CMPL    CLUB$L_LOCAL_CSB(R4),R2         ; Is this the local node?
                                0422   971          BEQL    50$                             ; If EQL yes, don't use it
                                0424   972          INCL    R0                              ; Else, say "CSB found"
                                0426   973          DECW    R1                              ; Update index for next time
                                0428   974          BRB     60$                             ; Exit loop
                                042A   975 50$:      SOBGTR  R1,30$                          ; Still in the vector? Continue.
                                042D   976          ;                                       ;
                                042D   977 60$:      MOVW    R1,ACB$W_LAST_INX(R9)           ; Update index for next time
                                0431   978          ;                                       ;
                                0431   979          ENBINT                                  ; Done with the vector
                                0434   980          RSB                                     ; Return
                                0435   981
```

```
                                0435   983 .SBTTL  'EXE$ALLOC_CSD  - Allocate and initialize a CSD block'
                                0435   984 ;++
                                0435   985 ;
                                0435   986 ;  Allocate and initialize fixed portions of CSD structure and an ACB to be
                                0435   987 ;  used as an internal work block.
                                0435   988 ;
                                0435   989 ;  EXE$ALLOC_CSD   should be used to allocate all CSD's.
                                0435   990 ;  EXE$DEALLOC_CSD should be used to deallocate all CSD's.
                                0435   991 ;
                                0435   992 ;  Because some fields in the CSD need reinitializing, and since the call to
                                0435   993 ;  EXE$DEALLOC_CSD is merely a request (the actual deallocation can only happen
                                0435   994 ;  when the CSD "runs down"), CSD's should not be recycled by the clients, but
                                0435   995 ;  rather a fresh one should be allocated for each use.
                                0435   996 ;
                                0435   997 ;
                                0435   998 ;  CALLING SEQUENCE:    JSB     EXE$ALLOC_CSD  at IPL 0
                                0435   999 ;
                                0435  1000 ;  INPUTS:       R2      Scratch
                                0435  1001 ;                R1      Size of structure to allocate (minimum CSD$AB_DATA)
                                0435  1002 ;                R0      Scratch
                                0435  1003 ;
                                0435  1004 ;  OUTPUTS:      R2      Address of allocated structure
                                0435  1005 ;                R1      Size allocated
                                0435  1006 ;                R0      Completion status:
                                0435  1007 ;                                SS$_NORMAL      => normal success
                                0435  1008 ;                                Low bit clear   => no buffer allocated
                                0435  1009 ;
                                0435  1010 ;--
                                0435  1011 EXE$ALLOC_CSD::
                          50    0435  1012         MOVL    S^#SS$_BADPARAM,R0                      ; Assume error
                                0438  1013 5$:      SAVIPL                                          ; Push IPL
                          8E    043B  1014         TSTL    (SP)+                                   ; Was is 0 ?
                          01    043D  1015         BEQL    10$                                     ; If EQL, okay
                                043F  1016         RSB                                             ; Else illegal IPL
                                0440  1017                                                         ;
                          38    0440  1018 10$:     PUSHR   #^M<R3,R4,R5>                           ; Save critical regs
                                0442  1019         ;
                                0442  1020         ;
                                0442  1021         ;   Check BYTCNT quota, wait if necessary.  The ACB is allocated along
                                0442  1022         ;   with the CSD block for simplicity.  BYTCNT quota is decremented for
                                0442  1023         ;   the ACB in order to prevent a process from gobbling up too much
                                0442  1024         ;   pool in case the CSD is small.
                                0442  1025         ;
                                0442  1026         ;
  00000052 8F                   0442  1027         CMPL    R1,#CSD$AB_DATA                         ; Is the request large enough ?
                          4A    0449  1028         BLSSU   60$                                     ; If LSSU, no
                       51 34    044B  1029         ADDL    #ACB$K_CSPLNG,R1                        ; Add in ACB size
                54 00000000'GF  044E  1030         MOVL    G^CTL$GL_PCB,R4                         ; Get address of PCB
                00000000'GF 16  0455  1031         JSB     G^EXE$BUFQUOPRC                         ; Wait for adequate BYTCNT quota
                      2E 50 E9  045B  1032         BLBC    R0,50$                                  ; If LBC, not enough
                                045E  1033         ;
                                045E  1034         ;
                                045E  1035         ;   EXE$BUFQUOPRC put us at IPL$_ASTDEL to prevent AST's from consuming
                                045E  1036         ;   any quota from the JIB.  Take the quota and restore IPL to 0 to
                                045E  1037         ;   allow the call to EXE$ALLOCBUF to wait if needed without blocking
                                045E  1038         ;   AST delivery (AST's may cause memory to be returned to pool) and
                                045E  1039         ;   hence avoiding a deadlock.  There is no need to stay at IPL$_ASTDEL
```

```
                     045E  1040          ;    to avoid process deallocation since we have not yet allocated any
                     045E  1041          ;    system wide resources (such as pool).
                     045E  1042          ;
                     045E  1043          ;
50   0080 C4   D0    045E  1044          MOVL    PCB$L_JIB(R4),R0                ; Get JIB
  20 A0   51   C2    0463  1045          SUBL    R1,JIB$L_BYTCNT(R0)             ; Take the quota
                     0467  1046          ;
          12   BB    0467  1047  30$:    PUSHR   #^M<R1,R4>                      ; Save quota taken, PCB
 00000000'GF   16    0469  1048          JSB     G^EXE$ALLOCBUF                  ; Allocate the buffer
                     046F  1049          ;                                       ...return at IPL$_ASTDEL (2)
          12   BA    046F  1050          POPR    #^M<R1,R4>                      ; Restore requested size, PCB
                     0471  1051          ;
        25 50   E8    0471  1052          BLBS    R0,80$                          ; If LBS, successful allocation
13 24 A4   0A   E0    0474  1053          BBS     #PCB$V_SSRWAIT,PCB$L_STS(R4),50$; If BS, wait mode DISABLED
      50   03   D0    0479  1054          MOVL    #RSN$_RPDYNMEM,R0               ; Resource to wait for
                     047C  1055          ;
                     047C  1056          SETIPL  #IPL$_SYNCH                     ; SCH$RWAIT requires this
          7E   DC    047F  1057          MOVPSL  -(SP)                           ; PSL onto stack for SCH$RWAIT
 00000000'GF   16    0481  1058          JSB     G^SCH$RWAIT                     ; Wait for resource
                     0487  1059          SETIPL  #0                              ; Restore IPL
                     048A  1060          ;
          DB   11    048A  1061          BRB     30$                             ; Loop
                     048C  1062  50$:    ;
                     048C  1063          ;
                     048C  1064          ;    Error return
                     048C  1065          ;
                     048C  1066          ;
52   0080 C4   D0    048C  1067          MOVL    PCB$L_JIB(R4),R2                ; Get JIB
  20 A2   51   C0    0491  1068          ADDL    R1,JIB$L_BYTCNT(R2)             ; Restore the quota taken
          52   D4    0495  1069  60$:    CLRL    R2                              ; Invalidate buffer pointer
          6D   11    0497  1070  70$:    BRB     100$                            ; Take common exit
                     0499  1071  80$:    ;
                     0499  1072          ;
                     0499  1073          ;    Got a buffer. Initialize the fixed portions.
                     0499  1074          ;
                     0499  1075          ;
          3E   BB    0499  1076          PUSHR   #^M<R1,R2,R3,R4,R5>             ; Protect volatile registers
00   6E   00   2C    049B  1077          MOVC5   #0,(SP),#0,-                    ; Clear the front end
62   0086 8F         049F  1078                  #ACB$K_CSPLNG+CSD$AB_DATA,(R2)  ;
          3E   BA    04A3  1079          POPR    #^M<R1,R2,R3,R4,R5>             ; Restore
                     04A5  1080          ;
                     04A5  1081          ;
                     04A5  1082          ;    Fill in the ACB fields as appropriate.  ACB$B_RMOD and ACB$L_PID
                     04A5  1083          ;    must be filled in just prior to queuing the AST since it may be
                     04A5  1084          ;    used as a fork block until then.
                     04A5  1085          ;
                     04A5  1086          ;    The PID must be saved in the ACB since it may not be trusted if
                     04A5  1087          ;    saved only in the CSD, especially if the CSD is to recieve a block
                     04A5  1088          ;    transfer.
                     04A5  1089          ;
                     04A5  1090          ;
                     04A5  1091          ASSUME  FKB$B_FIPL  EQ  ACB$B_RMOD
                     04A5  1092          ASSUME  FKB$L_FPC   EQ  ACB$L_PID
                     04A5  1093          ;
08 A2   51   B0      04A5  1094          MOVW    R1,             ACB$W_SIZE(R2)  ; Setup total size
0A A2   02   90      04A9  1095          MOVB    #DYN$C_ACB,     ACB$B_TYPE(R2)  ; Setup block type
32 A2   04   B0      04AD  1096          MOVW    #ACB$K_RETRY, ACB$W_RETRY(R2)   ; Setup retry count
```

```
                30 A2    00    90  04B1  1097          MOVB    #CEV$K_STA_I, ACB$B_STA(R2)     ; Initialize ACB state
          18 A2    05C4'CF    9E  04B5  1098          MOVAB   W^KAST,       ACB$L_KAST(R2)    ; Setup special-kernel AST ptr
          10 A2    05CE'CF    9E  04BB  1099          MOVAB   W^AST,        ACB$L_AST(R2)     ; Setup normal kernel AST ptr
            24 A2      60 A4    D0  04C1  1100          MOVL    PCB$L_PID(R4),ACB$L_USER_PID(R2); Copy internal PID
                       20 A2    D4  04C6  1101          CLRL                  ACB$L_USER_AST(R2); Zero user's AST address
            14 A2      34 A2    9E  04C9  1102          MOVAB   ACB$K_CSPLNG(R2),ACB$L_ASTPRM(R2) ; CSD address is AST parameter
                                    04CE  1103          ;
                    52   34    C0  04CE  1104          ADDL    #ACB$K_CSPLNG,R2                ; Advance to the CSD structure
                    51   34    C2  04D1  1105          SUBL    #ACB$K_CSPLNG,R1                ; Reduce size appropriately
                                    04D4  1106
                                    04D4  1107          ASSUME  CSD$B_SUBTYPE  EQ  1+CSD$B_TYPE
                                    04D4  1108
          0A A2    6465 8F    B0  04D4  1109          MOVW    #<DYN$C_CSD@8>!-                ; Fill in type/subtype
                                    04DA  1110            DYN$C_CLU,CSD$B_TYPE(R2)              ;
                08 A2    51    B0  04DA  1111          MOVW    R1,CSD$W_SIZE(R2)               ; Save allocation size
                                    04DE  1112
          42 A2    0084 C4    7D  04DE  1113          MOVQ    PCB$Q_PRIV(R4),CSD$Q_PROCPRIV(R2) ; Copy privileges
          4A A2    00BC C4    D0  04E4  1114          MOVL    PCB$L_UIC(R4), CSD$L_PROCUIC(R2)  ; Copy UIC
            36 A2      60 A4    D0  04EA  1115          MOVL    PCB$L_PID(R4), CSD$L_IPID(R2)     ; Copy internal PID
          50    00000000'GF    D0  04EF  1116          MOVL    G^CTL$GL_PHD,R0                 ; Get address of header
          4E A2    00F4 C0    D0  04F6  1117          MOVL    PHD$L_IMGCNT(R0),CSD$L_IMGCNT(R2) ; Copy image activation count
                                    04FC  1118
                54    CC A2    9E  04FC  1119          MOVAB   -ACB$K_CSPLNG(R2),R4            ; Get ACB address
                      0174     30  0500  1120          BSBW    ACT_INSQUE                      ; Queue ACB to 'idle' queue
                    50   01    D0  0503  1121          MOVL    #SS$_NORMAL,R0                  ; Success
                                    0506  1122          ;
                                    0506  1123          ;    That's it.
                                    0506  1124          ;
                                    0506  1125 100$:    SETIPL  #0                              ; Restore IPL
                        38    BA  0509  1126          POPR    #^M<R3,R4,R5>                   ; Restore regs
                        05    05  050B  1127          RSB                                     ; Done
                                    050C  1128
                                    050C  1129
```

```
                     050C  1131 .SBTTL  'EXE$DEALLOC_CSD  Deallocate CSD or mark it for deletion'
                     050C  1132 ;++
                     050C  1133 ;
                     050C  1134 ;  Deallocate CSD structure.   The deallocation is done via the PROC_EVENT
                     050C  1135 ;  mechanism to protect against deallocating the CSD if it active on some
                     050C  1136 ;  queue or there is a transfer in progress (there is no cancel request as
                     050C  1137 ;  part of the ACKMSG services).  Depending upon the current state, the CSD
                     050C  1138 ;  is either deallocated immediately or marked for delete when the CSD becomes
                     050C  1139 ;  free.
                     050C  1140 ;
                     050C  1141 ;  EXE$ALLOC_CSD   should be used to allocate all CSD's.
                     050C  1142 ;  EXE$DEALLOC_CSD should be used to deallocate all CSD's.
                     050C  1143 ;
                     050C  1144 ;  Because some fields in the CSD need reinitializing, and since the call to
                     050C  1145 ;  EXE$DEALLOC_CSD is merely a request (the actual deallocation can only happen
                     050C  1146 ;  when the CSD "runs down"), CSD's should not be recycled by the clients, but
                     050C  1147 ;  rather a fresh one should be allocated for each use.
                     050C  1148 ;
                     050C  1149 ;
                     050C  1150 ;  CALLING SEQUENCE:    JSB     EXE$DEALLOC_CSD at IPL 0 or 2.
                     050C  1151 ;
                     050C  1152 ;  INPUTS:       R0      Address of CSD to deallocate
                     050C  1153 ;                         CSD$W_SIZE(R0) = size of CSD
                     050C  1154 ;
                     050C  1155 ;  OUTPUTS:      R0-R3   Clobbered
                     050C  1156 ;
                     050C  1157 ;
                     050C  1158 ;--
                     050C  1159 EXE$DEALLOC_CSD::
           30   BB   050C  1160          PUSHR   #^M<R4,R5>                      ; Save regs
                     050E  1161          ;
     54    CC A0   9E 050E  1162          MOVAB   -ACB$K_CSPLNG(R0),R4            ; Get ACB block
     51      03   9A 0512  1163          MOVZBL  #CEV$_REQ_DEALL,R1              ; Setup event code
          0110   30 0515  1164          BSBW    PROC_EVENT                      ; Process the event
                     0518  1165          ;
           30   BA 0518  1166          POPR    #^M<R4,R5>                      ; Restore regs
     50    01   D0 051A  1167          MOVL    S^#SS$_NORMAL,R0                ; Setup return status
                05 051D  1168          RSB                                     ; Done
                   051E  1169
```

```
                     051E  1171 .SBTTL  'EXE$CSP_CALL   - Send a request message to local or remote CSP'
                     051E  1172 ;++
                     051E  1173 ;
                     051E  1174 ;  Call the Cluster Server Process on another node.
                     051E  1175 ;
                     051E  1176 ;  A block of data (the CSD) is sent to the CSP on the target node, and
                     051E  1177 ;  optionally recieve a response message into the same CSD.
                     051E  1178 ;
                     051E  1179 ;  If CSD$L_USER_AST is 0, then this routine does not return until the block
                     051E  1180 ;  transfer has completed, or has failed.
                     051E  1181 ;
                     051E  1182 ;  If CSD$L_USER_AST is non-zero, then this routine returns immediately.  If
                     051E  1183 ;  the return is with the low bit clear, then the AST will not be delivered and
                     051E  1184 ;  the CSD should be deallocated upon return.  If the return is with the low
                     051E  1185 ;  bit set in R0, then the AST routine should deallocate teh CSD.
                     051E  1186 ;
                     051E  1187 ;
                     051E  1188 ;
                     051E  1189 ;  R0's     AST          When to EXE$DEALLOC_CSD          When EXE$CSP_CALL
                     051E  1190 ;  low bit  specified    the CSD                          returns to caller
                     051E  1191 ;  -------  ---------    ------------------------         -----------------
                     051E  1192 ;    LBC      no         Upon return - no further         When the error is
                     051E  1193 ;                        AST's are delivered.             encountered.
                     051E  1194 ;    LBC      yes        Upon return - no further         When the error is
                     051E  1195 ;                        AST's are delivered.             encountered.
                     051E  1196 ;    LBS      no         Upon return                      When all dialogues
                     051E  1197 ;                                                         have completed.
                     051E  1198 ;    LBS      yes        By the AST routine or by         As soon as possible.
                     051E  1199 ;                        some action it schedules.
                     051E  1200 ;
                     051E  1201 ;
                     051E  1202 ;
                     051E  1203 ;  EXE$ALLOC_CSD   should be used to allocate all CSD's.
                     051E  1204 ;  EXE$DEALLOC_CSD should be used to deallocate all CSD's.
                     051E  1205 ;
                     051E  1206 ;  Because some fields in the CSD need reinitializing, and since the call to
                     051E  1207 ;  EXE$DEALLOC_CSD is merely a request (the actual deallocation can only happen
                     051E  1208 ;  when the CSD "runs down"), CSD's should not be recycled by the clients, but
                     051E  1209 ;  rather a fresh one should be allocated for each use.
                     051E  1210 ;
                     051E  1211 ;
                     051E  1212 ;  CALLING SEQUENCE:     JSB     EXE$CSP_CALL at IPL 0
                     051E  1213 ;
                     051E  1214 ;  INPUTS:       R2      Address of CSD structure
                     051E  1215 ;                R0      Scratch
                     051E  1216 ;
                     051E  1217 ;  OUTPUTS:      R0      SS$_... status code.
                     051E  1218 ;
                     051E  1219 ;                All other registers are preserved.
                     051E  1220 ;
                     051E  1221 ;--
                     051E  1222 EXE$CSP_CALL::                                       ; Send request to CSP
 007E 8F     BB      051E  1223         PUSHR   #^M<R1,R2,R3,R4,R5,R6>               ; Save volatile registers
                     0522  1224         .
   004F     30       0522  1225         BSBW    COMMON_SETUP                         ; Check IPL, get ACB, etc
 20 50      E9       0525  1226         BLBC    R0,200$                              ; If LBC, error
                     0528  1227         SETIPL  #IPL$_ASTDEL                         ; Go to IPL 2 to prevent AST's
```

```
                                052B  1228          ;
                                052B  1229          ;
                                052B  1230          ;   Request the start of the block transfer sequence
                                052B  1231          ;
                                052B  1232          ;
          51    02      D0      052B  1233          MOVL    #CEV$_REQ_BT,R1                 ; Event is 'request block xfer
          56    01      3C      052E  1234          MOVZWL  #SS$_NORMAL,R6                  ; Initialize status register
              00F4      30      0531  1235          BSBW    PROC_EVENT                      ; Process it
          50    56      D0      0534  1236          MOVL    R6,R0                           ; Pickup status
                                0537  1237          ;
                                0537  1238  100$:    SETIPL  #0                              ; Return to IPL 0
       09 31 A4    03      E0  053A  1239          BBS     #ACB$V_STS_BCST,ACB$B_STS(R4),200$ ; If BS, part of "broadcast"
             06    50      E9  053F  1240          BLBC    R0,200$                         ; If error, then return now
       52    34    A4      9E  0542  1241          MOVAB   ACB$K_CSPLNG(R4),R2             ; Pickup CSD
                 05          10  0546  1242          BSBB    WAIT                            ; Wait if necessary
                                0548  1243          ;
              007E  8F      BA  0548  1244  200$:    POPR    #^M<R1,R2,R3,R4,R5,R6>          ; Restore volatile registers
                    05      05  054C  1245  300$:    RSB                                     ; Return to caller
                                054D  1246
                                054D  1247
                                054D  1248  WAIT:    ;
                                054D  1249          ;
                                054D  1250          ;   We are waiting here for the block transfer to complete so that
                                054D  1251          ;   we can return to the user.  This is done whenever CSD$L_USER_AST
                                054D  1252          ;   is 0.  It allows a synchronous return.
                                054D  1253          ;
                                054D  1254          ;
                                054D  1255          ;   Inputs:     R4      Scratch
                                054D  1256          ;               R2      CSD address
                                054D  1257          ;               R0      SS$_NORMAL
                                054D  1258          ;
                                054D  1259          ;   Outputs:    R4      Garbage
                                054D  1260          ;
                                054D  1261          ;               All other registers are preserved
                                054D  1262          ;
                50    01      3C  054D  1263          MOVZWL  #SS$_NORMAL,R0                  ; Setup return status
             54     CC      A2  9E  0550  1264          MOVAB   -ACB$K_CSPLNG(R2),R4            ; Get ACB
       1A 31 A4      02      E1  0554  1265          BBC     #ACB$V_STS_WAIT,ACB$B_STS(R4),100$ ; If BC, not suspended
    54    00000000'GF      D0  0559  1266          MOVL    G^CTL$GL_PCB,R4                 ; Get PCB
                50    01      D0  0560  1267          MOVL    #RSN$_ASTWAIT,R0                ; Setup wait condition
                                0563  1268          ;
                                0563  1269          SETIPL  #IPL$_SYNCH                     ; SCH$RWAIT requires this
                      7E      DC  0566  1270          MOVPSL  -(SP)                           ; Put PSL on the stack
                00000000'GF   16  0568  1271          JSB     G^SCH$RWAIT                     ; Wait for resource
                                056E  1272          SETIPL  #0                              ; Restore IPL
                                0571  1273          ;
                      DA        11  0571  1274          BRB     WAIT                            ; Loop
                                05  0573  1275  100$:    RSB                                     ; Done
                                0574  1276
                                0574  1277
```

```
                                0574  1279 COMMON_SETUP:
                50    14   D0   0574  1280          MOVL    #SS$_BADPARAM,R0                        ; Assume error
                                0577  1281          SAVIPL                                          ; Push IPL
                      8E   D5   057A  1282          TSTL    (SP)+                                   ; Was is 0 ?
                      45   12   057C  1283          BNEQ    100$                                    ; If NEQ, illegal IPL
                                057E  1284          ;
                                057E  1285          ;
                                057E  1286          ;    Sanity check various fields in the CSD
                                057E  1287          ;
                                057E  1288          ;
                                057E  1289          ASSUME  CSD$B_SUBTYPE  EQ  1+CSD$B_TYPE
                                057E  1290
          0A A2   6465 8F   B1   057E  1291          CMPW    #<DYN$C_CSD@8>!DYN$C_CLU,CSD$B_TYPE(R2) ; Right structure?
                      3D   12   0584  1292          BNEQ    100$                                    ; If NEQ, return error
                54    CC   A2   9E   0586  1293          MOVAB   -ACB$K_CSPLNG(R2),R4                    ; Pickup ACB address
                      3A   A2   7C   058A  1294          CLRQ    CSD$Q_INT_IOSB(R2)                      ; Zero initial status
                53    08   A4   3C   058D  1295          MOVZWL  ACB$W_SIZE(R4),R3                       ; Get ACB total size
                53    54   C0   0591  1296          ADDL    R4,R3                                   ; Calculate end
          51    52    16   A2   C1   0594  1297          ADDL3   CSD$L_SENDOFF(R2),R2,R1                 ; Get begining of region
                51    12   A2   C0   0599  1298          ADDL    CSD$L_SENDLEN(R2),R1                    ; Calc end of region
                53    51   D1   059D  1299          CMPL    R1,R3                                   ; Within bounds ?
                      21   1A   05A0  1300          BGTRU   100$                                    ; If GTRU, out of bounds
          51    52    1E   A2   C1   05A2  1301          ADDL3   CSD$L_RECVOFF(R2),R2,R1                 ; Get begining of region
                51    1A   A2   C0   05A7  1302          ADDL    CSD$L_RECVLEN(R2),R1                    ; Calc end of region
                53    51   D1   05AB  1303          CMPL    R1,R3                                   ; Within bounds ?
                      13   1A   05AE  1304          BGTRU   100$                                    ; If GTRU, out of bounds
                                05B0  1305          ;
                                05B0  1306          ;
                                05B0  1307          ;    If the user want's an AST, let him have it without decrementing
                                05B0  1308          ;    its AST quota.  The ACB is needed anyway as a work block, and
                                05B0  1309          ;    the user has been charged for it via JIB$L_BYTCNT.
                                05B0  1310          ;
                                05B0  1311          ;
    20 A4     22 A2   D0   05B0  1312          MOVL    CSD$A_ASTADR(R2),ACB$L_USER_AST(R4)     ; Save user AST address
                      09   12   05B5  1313          BNEQ    70$                                     ; If NEQ, continue
    04 31 A4          03   E0   05B7  1314          BBS     #ACB$V_STS_BCST,ACB$B_STS(R4),70$       ; If BCST, never wait
       31 A4          04   88   05BC  1315          BISB    #ACB$M_STS_WAIT,ACB$B_STS(R4)           ; Else, wait until done
             50       01   D0   05C0  1316 70$:     MOVL    #SS$_NORMAL,R0                          ; Say "success"
                           05   05C3  1317 100$:     RSB                                             ; Done
                                05C4  1318
```

```
                        05C4  1320 .SBTTL  'KAST           - Special Kernel AST entry point'
                        05C4  1321 .SBTTL  'AST            - Normal  Kernel AST entry point'
                        05C4  1322 ;++
                        05C4  1323 ;
                        05C4  1324 ;  The proper event is determined and the event processor is called.
                        05C4  1325 ;
                        05C4  1326 ;--
                        05C4  1327 KAST:                                            ; Special Kernel AST
                        05C4  1328 ;
                        05C4  1329 ;
                        05C4  1330 ;       The ACB is in R5.  IPL is IPL$_ASTDEL (2).
                        05C4  1331 ;
                        05C4  1332 ;       R0 thru R5 may be clobbered upon return to caller
                        05C4  1333 ;
                        05C4  1334 ;
    52    14 A5   D0    05C4  1335         MOVL    ACB$L_ASTPRM(R5),R2              ; Get CSD
      51    0C   D0    05C8  1336         MOVL    #CEV$_KAST_DEL,R1                ; Setup event code
            1A   10    05CB  1337         BSBB    ASTEVT                           ; Process event
            05   05    05CD  1338         RSB                                      ; Done
                        05CE  1339
                        05CE  1340 AST:                                             ; Normal Kernel AST
                        05CE  1341 ;
                        05CE  1342 ;
                        05CE  1343 ;       The ACB is the AST parameter.  IPL is 0.
                        05CE  1344 ;
                        05CE  1345 ;       All regs but R0,R1 must be saved/restored.
                        05CE  1346 ;
                        05CE  1347 ;
              003C      05CE  1348         .WORD   ^M<R2,R3,R4,R5>                  ; Entry mask
    52    04 AC   D0    05D0  1349         MOVL    4(AP),R2                         ; Get CSD address
      51    0D   D0    05D4  1350         MOVL    #CEV$_AST_DEL,R1                 ; Setup event code
            0E   10    05D7  1351         BSBB    ASTEVT                           ; Do AST common processing
            54   D5    05D9  1352         TSTL    R4                               ; Still have an ACB ?
            09   13    05DB  1353         BEQL    30$                              ; If EQL, no
    50    20 A4   D0    05DD  1354         MOVL    ACB$L_USER_AST(R4),R0            ; Get AST address
            03   13    05E1  1355         BEQL    30$                              ; If EQL, none
       60   6C   FA    05E3  1356         CALLG   (AP),(R0)                        ; Call the user AST routine
            04   05    05E6  1357 30$:    RET                                      ; Done
                        05E7  1358
    54    CC A2   9E    05E7  1359 ASTEVT: MOVAB   -ACB$K_CSPLNG(R2),R4             ; Get ACB address
 1B 31 A4     01   E5    05EB  1360         BBCC    #ACB$V_STS_QUE,ACB$B_STS(R4),90$; ACB no longer queued to PCB
          20 A4   D5    05F0  1361         TSTL    ACB$L_USER_AST(R4)               ; Does user want AST delivered?
             0F   13    05F3  1362         BEQL    50$                              ; If EQL, no
50    00000000'GF   D0    05F5  1363         MOVL    G^CTL$GL_PHD,R0                  ; Get current PHD
 4E A2    00F4 C0    D1    05FC  1364         CMPL    PHD$L_IMGCNT(R0),CSD$L_IMGCNT(R2) ; Compare image deactivations
             03   13    0602  1365         BEQL    70$                              ; If EQL, same image is running
          51   0E   D0    0604  1366 50$:    MOVL    #CEV$_NO_AST,R1                  ; No user AST to deliver
              001E   30    0607  1367 70$:    BSBW    PROC_EVENT                       ; Process the event
                     05    060A  1368         RSB                                      ; Done
                        060B  1369
                        060B  1370 90$:    BUG_CHECK  INCONSTATE,FATAL              ; Queued state is inconsistent
                        060F  1371
                        060F  1372
```

```
                        060F  1374 .SBTTL  'PROC_EVENT_ASY - Process CSD event if process is still around'
                        060F  1375 .SBTTL  'PROC_EVENT     - Process CSD event'
                        060F  1376 ;+
                        060F  1377 ;
                        060F  1378 ;  This routine processes all CSD events and is state table driven.  Action
                        060F  1379 ;  routines are called until the null event is detected.  Each action routine
                        060F  1380 ;  generates a new event, which it returns in R1, and returns with the low bit
                        060F  1381 ;  set in R0 only if the indicated state change is to be performed.
                        060F  1382 ;
                        060F  1383 ;
                        060F  1384 ;  CALLING SEQUENCE:    JSB PROC_EVENT at IPL$_SYNCH or lower
                        060F  1385 ;
                        060F  1386 ;  INPUTS:       R5      Scratch
                        060F  1387 ;                R4      ACB ptr
                        060F  1388 ;                R3      Scratch
                        060F  1389 ;                R2      Optional event parameter
                        060F  1390 ;                R1      Standard event longword
                        060F  1391 ;                R0      Scratch
                        060F  1392 ;
                        060F  1393 ;                All other registers are scratch.
                        060F  1394 ;
                        060F  1395 ;  OUTPUTS:      R4      Unchanged, or zero if deallocated
                        060F  1396 ;
                        060F  1397 ;                All other registers between R0 and R5 are clobbered
                        060F  1398 ;
                        060F  1399 ;-
                        060F  1400 PROC_EVENT_ASY:                                  ; Process asynch event
      50    24 A4   3C  060F  1401          MOVZWL  ACB$L_USER_PID(R4),R0         ; Get process index
52    00000000'GF   D0  0613  1402          MOVL    G^SCH$GL_PCBVEC,R2            ; Get address of PCB vector
      52    6240    D0  061A  1403          MOVL    (R2)[R0],R2                   ; Get PCB itself
   60 A2    24 A4   D1  061E  1404          CMPL    ACB$L_USER_PID(R4),PCB$L_PID(R2); Is this process still here?
            03      13  0623  1405          BEQL    PROC_EVENT                    ; If EQL, yes
          021C      31  0625  1406          BRW     DEALC_CSD                     ; Else, deallocate CSD/ACB
                        0628  1407 ;
                        0628  1408 PROC_EVENT:                                    ; Process all CSD events
                        0628  1409          ASSUME  IPL$_SYNCH  EQ  IPL$_SCS
                        0628  1410          DSBINT  #IPL$_SYNCH                   ; Synchronize
                        062E  1411 10$:     ;
                        062E  1412          ;
                        062E  1413          ;    Find appropriate state table entry
                        062E  1414          ;
                        062E  1415          ;
        51    0F    D1  062E  1416          CMPL    S^#CEV$_MAX_EVT,R1            ; Is event within range ?
              40    1F  0631  1417          BLSSU   200$                          ; If LSSU then bug exists
   50   51    06    C5  0633  1418          MULL3   S^#CEV$K_STATES,R1,R0         ; Bias for current event
      53    30 A4   9A  0637  1419          MOVZBL  ACB$B_STA(R4),R3              ; Get ACB state
         50    53    C0  063B  1420          ADDL    R3,R0                         ; Add current state offset
53    FA51 CF40    3E  063E  1421          MOVAW   W^CEV$AW_STA_TAB[R0],R3       ; Address state table entry
                        0644  1422          ;
                        0644  1423          ;
                        0644  1424          ;
                        0644  1425          ;    Dispatch to the action routine with the following:
                        0644  1426          ;
                        0644  1427          ;    INPUTS:     R5      Scratch
                        0644  1428          ;                R4      ACB pointer
                        0644  1429          ;                R3      CSID of target system
                        0644  1430          ;                R2      CSD pointer
```

```
                                0644  1431          ;                       R1      Event code
                                0644  1432          ;                       R0      Scratch
                                0644  1433          ;
                                0644  1434          ;       ON RETURN:      R5      Garbage
                                0644  1435          ;                       R4      ACB pointer
                                0644  1436          ;                       R3      Garbage
                                0644  1437          ;                       R2      Garbage
                                0644  1438          ;                       R1      Next event code to chain to
                                0644  1439          ;                       R0      Low bit set to request state change
                                0644  1440          ;                               Low bit clear to inhibit state change
                                0644  1441          ;
                                0644  1442          ;
                          83    9F  0644  1443          PUSHAB  (R3)+                   ; Save table address
                      53  63    9A  0646  1444          MOVZBL  (R3),R3                 ; Get action routine index
                50  F983  CF    9E  0649  1445          MOVAB   W^CEV$AL_ACTTAB,R0      ; Get action routine table
                  50   6043    C0  064E  1446          ADDL    (R0)[R3],R0             ; Get action routine address
                52   14  A4    D0  0652  1447          MOVL    ACB$L_ASTPRM(R4),R2     ; Get the CSD
                53   0E  A2    D0  0656  1448          MOVL    CSD$L_CSID(R2),R3       ; Get the CSID
                          60    16  065A  1449          JSB     (R0)                    ; Dispatch
                          53  8ED0  065C  1450          POPL    R3                      ; Get next state, cleanup stack
                          54    D5  065F  1451          TSTL    R4                      ; Is ACB still there ?
                          0C    13  0661  1452          BEQL    100$                    ; If EQL, its been deallocated
                      04  50    E9  0663  1453          BLBC    R0,50$                  ; Avoid state change if LBC
                30  A4    63    90  0666  1454          MOVB    (R3),ACB$B_STA(R4)      ; Change state
                      51    00    D1  066A  1455  50$:    CMPL    S^#CEV$_EXIT,R1         ; Are we done ?
                          BF    12  066D  1456          BNEQ    10$                     ; If NEQ then process next event
                                066F  1457          ;
                                066F  1458  100$:   ENBINT                              ; Restore IPL
                          05    0672  1459          RSB                               ; Done
                                0673  1460          ;
                                0673  1461  200$:   BUG_CHECK  INCONSTATE,FATAL       ; Signal the bug
                                0677  1462          ;
                                0677  1463          ;
```

```
                     0677  1465 .SBTTL  'ACT_INSQUE     - Queue  ACB to CSP$Q_ACB_IDLE'
                     0677  1466 .SBTTL  'ACT_REMQUE     - Remove ACB from current (internal) queue'
                     0677  1467 ;+
                     0677  1468 ;
                     0677  1469 ;  The ACB queue operation is performed.  Upon return, the event code passed
                     0677  1470 ;  in R1 is unchanged and the low bit of R0 is set.  This will force the same
                     0677  1471 ;  event to be reprocessed after the state change.
                     0677  1472 ;
                     0677  1473 ;
                     0677  1474 ;  INPUTS:      R4      ACB pointer
                     0677  1475 ;               R1      Event to be processed
                     0677  1476 ;               R0      Scratch
                     0677  1477 ;
                     0677  1478 ;  OUTPUTS:     R4      Unchanged
                     0677  1479 ;               R1      Unchanged
                     0677  1480 ;               R0      Low bit set to force state change
                     0677  1481 ;
                     0677  1482 ;-
                     0677  1483 ACT_INSQUE:                                          ; Put ACB on 'idle' queue
09 31 A4   01   E2   0677  1484         BBSS    #ACB$V_STS_QUE,ACB$B_STS(R4),10$;  Mark ACB as 'queued'
 FADF CF   64   0E   067C  1485         INSQUE  (R4),CSP$Q_ACB_IDLE                ; Remove from current queue
       50   01   D0 0681  1486         MOVL    #1,R0                              ; Request state change
            05   0684  1487         RSB                                        ; Return to reprocess same event
                 0685  1488
                 0685  1489 10$:     BUG_CHECK  INCONSTATE,FATAL                ; Queued state is inconsistent
                 0689  1490
                 0689  1491 ACT_REMQUE:                                          ; Dequeue ACB and deallocate it
07 31 A4   01   E5   0689  1492         BBCC    #ACB$V_STS_QUE,ACB$B_STS(R4),10$;  Mark ACB as 'not queued'
       54   64   0F 068E  1493         REMQUE  (R4),R4                            ; Remove from current queue
       50   01   D0 0691  1494         MOVL    #1,R0                              ; Request state change
            05   0694  1495         RSB                                        ; Return to reprocess same event
                 0695  1496
                 0695  1497 10$:     BUG_CHECK  INCONSTATE,FATAL                ; Queued state is inconsistent
                 0699  1498
```

```
                                                     J 14
                     0699  1500 .SBTTL  'ACT_GET_CDRP   - Allocate a warm CDRP for block transfer'
                     0699  1501 ;+
                     0699  1502 ;
                     0699  1503 ;  INPUTS:       R5      Scratch
                     0699  1504 ;                R4      ACB pointer
                     0699  1505 ;                R3      CSID of target system
                     0699  1506 ;                R2      CSD pointer
                     0699  1507 ;                R1      Scratch
                     0699  1508 ;                R0      Scratch
                     0699  1509 ;
                     0699  1510 ;  OUTPUTS:      R5      CDRP pointer if allocation was a success
                     0699  1511 ;                R4      ACB pointer
                     0699  1512 ;                R3      Garbage
                     0699  1513 ;                R2      Garbage
                     0699  1514 ;                R1      CEV$_NO_CDRP    if no CDRP was available
                     0699  1515 ;                        CEV$_GOT_CDRP   if CDRP allocation was successful
                     0699  1516 ;                R0      Low bit set to request state change
                     0699  1517 ;-
                     0699  1518
                     0699  1519 ACT_GET_CDRP:                                    ; Allocate warm CDRP
                     0699  1520         ;
                     0699  1521         ;
                     0699  1522         ;       Allocate a warm CDRP and fill it in as appropriate
                     0699  1523         ;
                     0699  1524         ;
00000000'GF   16     0699  1525         JSB     G^CNX$ALLOC_WARMCDRP                ; Get the CDRP
      51    04     9A 069F  1526         MOVZBL  #CEV$_NO_CDRP,R1                    ; Assume allocation failure
         31 50     E9 06A2  1527         BLBC    R0,100$                             ; If LBC, allocation failed
     52    14 A4    D0 06A5  1528         MOVL    ACB$L_ASTPRM(R4),R2                 ; Get the CSD again
     3C A5    54    D0 06A9  1529         MOVL    R4,CDRP$L_VAL5(R5)                  ; Save ACB address
4C A5    06DA'CF    9E 06AD  1530         MOVAB   W^REQ_MSGBLD,CDRP$L_MSGBLD(R5)      ; Setup message build routine
        4A A5    94 06B3  1531         CLRB    CDRP$B_CNXRMOD(R5)                  ; Kernel mode
                    06B6  1532         ;
 50   00000000'GF   D0 06B6  1533         MOVL    G^MMG$GL_SPTBASE,R0                 ; Get SPT base address
   51   52    15    09 06BD  1534         EXTZV   #VA$V_VPN,#VA$S_VPN,R2,R1           ; Get page number
      40 A5    6041 DE 06C2  1535         MOVAL   (R0)[R1],CDRP$L_CNXSVAPTE(R5)       ; Store SVAPTE
   46 A5    08 A2 3C 06C7  1536         MOVZWL  CSD$W_SIZE(R2),CDRP$L_CNXBCNT(R5)   ; Store BCNT
44 A5   52   FE00 8F    AB 06CC  1537         BICW3   #^C<VA$M_BYTE>,R2,CDRP$W_CNXBOFF(R5) ; Store BOFF
                    06D3  1538         ;
                    06D3  1539         ;
                    06D3  1540         ;       Exit with proper new event code
                    06D3  1541         ;
                    06D3  1542         ;
            51    06  9A 06D3  1543         MOVZBL  #CEV$_GOT_CDRP,R1                   ; Setup new event
            50    01  D0 06D6  1544 100$:   MOVL    #1,R0                             ; Request state change
                      05 06D9  1545         RSB                                       ; Done
                    06DA  1546
                    06DA  1547 REQ_MSGBLD:
                    06DA  1548         ;
                    06DA  1549         ;       ACKMSG calls us here to build the request message.
                    06DA  1550         ;
                    06DA  1551         ;       INPUTS:     R5      CDRP ptr
                    06DA  1552         ;                   R4      PDT ptr
                    06DA  1553         ;                   R3      CSB ptr
                    06DA  1554         ;                   R2      Message pointer
                    06DA  1555         ;
                    06DA  1556         ;
```

```
                50    3C A5    D0  06DA  1557          MOVL    CDRP$L_VAL5(R5),R0                          ; Get ACB address
                50    14 A0    D0  06DE  1558          MOVL    ACB$L_ASTPRM(R0),R0                         ; Get the CSD again
          1C A2       08 A0    3C  06E2  1559          MOVZWL  CSD$W_SIZE(R0),CSPMSG$L_CSD_SIZE(R2)        ; Setup size
          1A A2       0C A0    B0  06E7  1560          MOVW    CSD$W_CODE(R0),CSPMSG$W_CLIENT(R2)          ; Setup client code
             08 A2        06    90  06EC  1561          MOVB    #CLSMSG$K_FAC_CSP,CLSMSG$B_FACILITY(R2)     ; Tell ACKMSG it's us
                09 A2     94  06F0  1562          CLRB     CLSMSG$B_FUNC(R2)                              ; Our func code
                                  06F3  1563                                                              ; - not used yet
                          05  06F3  1564          RSB                                                     ; Done
                              06F4  1565
```

```
                                06F4  1567 .SBTTL  'ACT_FORK_WAIT  - Fork and wait for up to 1 second'
                                06F4  1568 ;+
                                06F4  1569 ;
                                06F4  1570 ;  INPUTS:       R5      Scratch
                                06F4  1571 ;                R4      ACB pointer
                                06F4  1572 ;                R3      CSID of target system
                                06F4  1573 ;                R2      CSD pointer
                                06F4  1574 ;                R1      Scratch
                                06F4  1575 ;                R0      Scratch
                                06F4  1576 ;
                                06F4  1577 ;  OUTPUTS:      R5      CDRP pointer if allocation was a success
                                06F4  1578 ;                R4      ACB pointer
                                06F4  1579 ;                R3      Garbage
                                06F4  1580 ;                R2      Garbage
                                06F4  1581 ;                R1      CEV$_EXIT   if okay to retry
                                06F4  1582 ;                        CEV$_GIVEUP if retry count exceeded
                                06F4  1583 ;                R0      Low bit set to request state change
                                06F4  1584 ;
                                06F4  1585 ;  SIDE EFFECTS:        When the fork returns, PROC_EVENT is called with the
                                06F4  1586 ;                        event CEV$_FORK_DONE
                                06F4  1587 ;
                                06F4  1588 ;-
                                06F4  1589 ACT_FORK_WAIT:                                   ; Fork and wait for up to 1 sec.
51  0B   D0                    06F4  1590         MOVL    #CEV$_GIVE_UP,R1                 ; Assume retry count exceeded
  32 A4   B7                    06F7  1591         DECW    ACB$W_RETRY(R4)                  ; Account for retry
     13   15                    06FA  1592         BLEQ    30$                              ; If LEQ, count exceeded
                                06FC  1593
                                06FC  1594         ASSUME  FKB$B_FIPL EQ  ACB$B_RMOD
                                06FC  1595         ASSUME  FKB$L_FPC  EQ  ACB$L_PID
                                06FC  1596         ASSUME  FKB$L_FR3  EQ  ACB$L_AST
                                06FC  1597         ASSUME  FKB$L_FR4  EQ  ACB$L_ASTPRM
                                06FC  1598
   55   54   D0                  06FC  1599         MOVL    R4,R5                            ; Setup fork block address
53   10 A5   7D                  06FF  1600         MOVQ    FKB$L_FR3(R5),R3                 ; Get ACB fields to be saved
0B A5   08   90                  0703  1601         MOVB    #IPL$_SCS,FKB$B_FIPL(R5)         ; Setup fork IPL
        0A   10                  0707  1602         BSBB    50$                              ; Create fork thread
   54   55   D0                  0709  1603         MOVL    R5,R4                            ; Re-establish ACB pointer
   51   00   9A                  070C  1604         MOVZBL  #CEV$_EXIT,R1                    ; Setup next event code
   50   01   D0                  070F  1605 30$:    MOVL    #1,R0                            ; Request state change
             05                  0712  1606         RSB                                      ; Done
                                 0713  1607
15 31 A5   01   E2               0713  1608 50$:    BBSS    #ACB$V_STS_QUE,ACB$B_STS(R5),90$; Mark ACB as 'queued'
                                 0718  1609         FORK_WAIT                                ; Fork and wait for a second
0A 31 A5   01   E5               071E  1610         BBCC    #ACB$V_STS_QUE,ACB$B_STS(R5),90$; Mark ACB as 'not queued'
   54   55   D0                  0723  1611         MOVL    R5,R4                            ; Re-establish ACB pointer
   51   05   D0                  0726  1612         MOVL    #CEV$_FORK_DONE,R1               ; Setup event
          FEE3   30              0729  1613         BSBW    PROC_EVENT_ASY                   ; Process event if process is
                                 072C  1614                                                  ; still here, else deallocate
                                 072C  1615                                                  ; the ACB/CSD
             05                  072C  1616         RSB                                      ; Done
                                 072D  1617
                                 C72D  1618 90$:    BUG_CHECK  INCONSTATE,FATAL              ; Queued state is inconsistent
                                 0731  1619
```

```
                                0731  1621 .SBTTL  'ACT_REQ_ILL_BT - Request illegal block-transfer'
                                0731  1622 .SBTTL  'ACT_BLOCK_XFER - Request ACKMSG Block Transfer'
                                0731  1623 ;+
                                0731  1624 ;
                                0731  1625 ;  INPUTS:       R5      CDRP pointer
                                0731  1626 ;                 R4      ACB pointer
                                0731  1627 ;                 R3      CSID of target system
                                0731  1628 ;                 R2      CSD pointer
                                0731  1629 ;                 R1      Scratch
                                0731  1630 ;                 R0      Scratch
                                0731  1631 ;
                                0731  1632 ;  OUTPUTS:      R5      Garbage
                                0731  1633 ;                 R4      ACB pointer
                                0731  1634 ;                 R3      Garbage
                                0731  1635 ;                 R2      Garbage
                                0731  1636 ;                 R1      CEV$_EXIT
                                0731  1637 ;                         CEV$_BT_DONE
                                0731  1638 ;                         CEV$_CSP_BUSY
                                0731  1639 ;                 R0      Low bit set to request state change
                                0731  1640 ;
                                0731  1641 ;  SIDE EFFECTS:         When the fork returns, PROC_EVENT is called with the
                                0731  1642 ;                         event CEV$_FORK_DONE
                                0731  1643 ;
                                0731  1644 ;-
                                0731  1645 ACT_REQ_ILL_BT:                                  ; User requested block transfer
                                0731  1646                                                  ; with CSD in the wrong state
     56  000002C4 8F    D0  0731  1647          MOVL    #SS$_DEVACTIVE,R6                   ; Say 'CSD in wrong state'
           51    00    D0  0738  1648          MOVL    S^#CEV$_EXIT,R1                     ; No further events
           50    01    90  073B  1649          MOVB    #1,R0                               ; Allow state transition
                       05  073E  1650          RSB
                           073F  1651
                           073F  1652 ACT_BLOCK_XFER:                                      ; Request ACKMSG block transfer
                           073F  1653          ;
                           073F  1654          ;
                           073F  1655          ;   CNX$BLOCK_XFER usually  returns asynchronously.  Therefore, we
                           073F  1656          ;   must call a routine to call CNX$BLOCK_XFER so that we can return
                           073F  1657          ;   to our caller with the correct values in the registers.
                           073F  1658          ;
                           073F  1659          ;
   31 A4    01    88      073F  1660          BISB    #ACB$M_STS_ASY,ACB$B_STS(R4)        ; Mark ACB for asynch access
              54    DD    0743  1661          PUSHL   R4                                  ; Save ACB pointer
              13    10    0745  1662          BSBB    30$                                 ; Make request and return
              54  8ED0    0747  1663          POPL    R4                                  ; Restore ACB pointer
03 31 A4    00    E5      074A  1664          BBCC    #ACB$V_STS_ASY,ACB$B_STS(R4),10$    ; If BC, CNX$BLOCK_XFER returned
                           074F  1665                                                      ; synchronously.
      51    00    9A      074F  1666          MOVZBL  #CEV$_EXIT,R1                       ; No further events for now
      50    01    D0      0752  1667 10$:     MOVL    #1,R0                               ; Request state change
                 05      0755  1668          RSB                                         ; Done
                           0756  1669
                           0756  1670 20$:     BUG_CHECK  INCONSTATE,FATAL                 ; Queued state is inconsistent
                           075A  1671
                           075A  1672 30$:     ;
                           075A  1673          ;   Request block transfer.
                           075A  1674          ;
                           075A  1675          ;   We are resumed after the call to BLOCK_XFER when block transfer
                           075A  1676          ;   sequence has completed with the following registers setup:
                           075A  1677          ;
```

```
                                075A  1678          ;       R5      Address of CDRP
                                075A  1679          ;       R4      Address of PDT
                                075A  1680          ;       R3      CSB address
                                075A  1681          ;       R2      Address of response message buffer      (if R0 has LBS)
                                075A  1682          ;       R1      Scratch
                                075A  1683          ;       R0      Status
                                075A  1684          ;
                                075A  1685          ;
F3 31 A4    01    E2            075A  1686          BBSS    #ACB$V_STS_QUE,ACB$B_STS(R4),10$; Mark ACB as 'queued'
 FA04 CF    64    0E            075F  1687          INSQUE  (R4),CSP$Q_ACB_XFER             ; Queue to 'active xfer' queue
       FB99'    30            0764  1688          BSBW    CNX$BLOCK_XFER                  ; Do block transfer sequence
   54    3C A5    D0            0767  1689          MOVL    CDRP$L_VAL5(R5),R4              ; Get ACB pointer
E6 31 A4    01    E5            076B  1690          BBCC    #ACB$V_STS_QUE,ACB$B_STS(R4),20$; Mark ACB as 'not queued'
      54    64    0F            0770  1691          REMQUE  (R4),R4                         ; Remove from 'active xfer' list
                                0773  1692          ;
      0D 50    E8            0773  1693          BLBS    R0,50$                          ; If LBS, then no error
      51    08    D0            0776  1694          MOVL    #CSPMSG$K_RSP_SYNERR,R1         ; Assume synchronous error
09 31 A4    00    E0            0779  1695          BBS     #ACB$V_STS_ASY,ACB$B_STS(R4),60$; If BS, return was synchronous
      51    07    D0            077E  1696          MOVL    #CSPMSG$K_RSP_ASYNERR,R1        ; Asynchronous error
            04    11            0781  1697          BRB     60$                             ; Continue
   51    1B A2    9A            0783  1698 50$:     MOVZBL  CSPMSG$B_RSP(R2),R1             ; Get the response code
   6E A4    50    7D            0787  1699 60$:     MOVQ    R0,ACB$K_CSPLNG+CSD$Q_INT_IOSB(R4) ; Save status info
            1B    10            078B  1700          BSBB    DUMP_CDRP                       ; Dump CDRP using R0 status
   50    6E A4    7D            078D  1701          MOVQ    ACB$K_CSPLNG+CSD$Q_INT_IOSB(R4),R0 ; Recover status info
                                0791  1702          ;
                                0791  1703          ;
                                0791  1704          ;   If ACB$V_STS_ASY is still set then the return is synchronous and
                                0791  1705          ;   all we have to do, after clearing the flag, is to return and let
                                0791  1706          ;   our caller chain to the next event since we are still in the event
                                0791  1707          ;   processing loop.
                                0791  1708          ;
                                0791  1709          ;   Otherwise, we must call PROC_EVENT_ASY to check to see if the
                                0791  1710          ;   process is still there, and if so, to process the new event.
                                0791  1711          ;
                                0791  1712          ;
      09    51    D1            0791  1713          CMPL    R1,#CSPMSG$K_RSP_MAX            ; Within range ?
            03    1B            0794  1714          BLEQU   70$                             ; If LEQU, okay
      51    01    90            0796  1715          MOVB    #CSPMSG$K_RSP_ILL,R1            ; Override with our own code
51    F9B6 CF41    9A            0799  1716 70$:     MOVZBL  CEV$AB_RSP_CEV[R1],R1           ; Convert response to an event
 03 31 A4    00    E4            079F  1717          BBSC    #ACB$V_STS_ASY,ACB$B_STS(R4),90$; If BS, return was synchronous
         FE68    30            07A4  1718          BSBW    PROC_EVENT_ASY                  ; Process event
            05    07A7  1719 90$:     RSB                                             ; Return
                                07A8  1720          ;
                                07A8  1721          ;
                                07A8  1722 DUMP_CDRP:                                       ; Dump CDRP according to status
         03 50    E9            07A8  1723          BLBC    R0,10$                          ; If LBC, special cleanup
          F852'   31            07AB  1724          BRW     CNX$DEALL_WARMCDRP_CSB          ; Deallocate ACKMSG resources
                                07AE  1725 10$:     ;
                                07AE  1726          ;
                                07AE  1727          ;   The following code assumes that the CDRP is "cold", that is,
                                07AE  1728          ;   contains no associated buffer or RSPID.
                                07AE  1729          ;
                                07AE  1730          ;
      0C    BB                  07AE  1731          PUSHR   #^M<R2,R3>                      ; Save regs
   50    55    D0            07B0  1732          MOVL    R5,R0                           ; Get address for deallocation
         55    D4            07B3  1733          CLRL    R5                              ; CDRP is now gone
00000000'GF    16            07B5  1734          JSB     G^EXE$DEANONPAGED               ; Deallocate it
```

```
                07BB  1735
                07BB  1736
0C   BA         07BB  1736          POPR    #^M<R2,R3>                      ;
                07BB  1736                                                  ; Restore regs
05   05         07BD  1737          RSB                                     ; Done
                07BE  1738
```

```
                        07BE  1740 .SBTTL  'ACT_NO_AST     - No AST to deliver - deallocate CSD if broadcast'
                        07BE  1741 .SBTTL  'ACT_GIVE_UP    - Retry count has be exhausted, give up'
                        07BE  1742 .SBTTL  'ACT_QUE_KAST   - Queue Special Kernel AST to process'
                        07BE  1743 .SBTTL  'ACT_QUE_AST    - Queue Normal  Kernel AST to process'
                        07BE  1744 ;+
                        07BE  1745 ;
                        07BE  1746 ;  Come here when the Block transfer has completed or failed.
                        07BE  1747 ;
                        07BE  1748 ;
                        07BE  1749 ;  INPUTS:       R5      Scratch
                        07BE  1750 ;                R4      ACB pointer
                        07BE  1751 ;                R3      CSID of target system
                        07BE  1752 ;                R2      CSD pointer
                        07BE  1753 ;                R1      Scratch
                        07BE  1754 ;                R0      Scratch
                        07BE  1755 ;
                        07BE  1756 ;  OUTPUTS:      R5      Garbage
                        07BE  1757 ;                R4      ACB pointer
                        07BE  1758 ;                R3      Garbage
                        07BE  1759 ;                R2      Garbage
                        07BE  1760 ;                R1      CEV$_EXIT
                        07BE  1761 ;                R0      Low bit set to request state change
                        07BE  1762 ;
                        07BE  1763 ;-
                        07BE  1764                  .ENABL  LSB
                        07BE  1765 ACT_NO_AST:                                  ; No AST to deliver
31 A4   04   8A         07BE  1766                  BICB    #ACB$M_STS_WAIT,ACB$B_STS(R4)   ; No need to wait any longer
34 31 A4   03   E1      07C2  1767                  BBC     #ACB$V_STS_BCST,ACB$B_STS(R4),30$ ; If BC, not part of broadcast
      51   03   D0      07C7  1768                  MOVL    #CEV$_REQ_DEALL,R1      ; Else, request deallocation
           32   11      07CA  1769                  BRB     40$                     ; Continue
                        07CC  1770                                                  ;
                        07CC  1771 ACT_GIVE_UP:                                 ; Retry count exceeded
      022C 8F   3C      07CC  1772                  MOVZWL  #SS$_TIMEOUT,-          ; Setup status
         6E A4          07D0  1773                          ACB$R_CSPLNG+CSD$Q_INT_IOSB(R4)
2B 31 A4    01   E4      07D2  1774                  BBSC    #ACB$V_STS_QUE,ACB$B_STS(R4),50$; Make sure ACB is not queued
                        07D7  1775                                                  ;
                        07D7  1776 ACT_QUE_KAST:                                ; Queue Special Kernel AST
0C A4    24 A4   D0      07D7  1777                  MOVL    ACB$L_USER_PID(R4),ACB$L_PID(R4); Copy internal PID
08 A4    A0 8F   90      07DC  1778                  MOVB    #ACB$M_KAST!-           ; Mark as 'special kernel'
                        07E1  1779                           ACB$M_NODELETE,ACB$B_RMOD(R4) ; and don't delete ACB
      52   01   D0      07E1  1780                  MOVL    #PRI$_IOCOM,R2          ; Setup priority increment class
           02   11      07E4  1781                  BRB     10$                     ; Continue
                        07E6  1782                                                  ;
                        07E6  1783 ACT_QUE_AST:                                 ; Queue Normal Kernel AST
      52   D4           07E6  1784                  CLRL    R2                      ; Use null priority inc. class
15 31 A4   01   E2      07E8  1785 10$:             BBSS    #ACB$V_STS_QUE,ACB$B_STS(R4),50$; ACB will be queued to the PCB
      55   54   D0      07ED  1786                  MOVL    R4,R5                   ; Setup ACB pointer
           54   DD      07F0  1787                  PUSHL   R4                      ; Save ACB address
00000000'GF   16        07F2  1788                  JSB     G^SCH$QAST              ; Queue the AST
           54 8ED0      07F8  1789                  POPL    R4                      ; Restore ACB address
      51   00   D0      07FB  1790 30$:             MOVL    #CEV$_EXIT,R1           ; No new events
      50   01   D0      07FE  1791 40$:             MOVL    #1,R0                   ; Request state change
                 05     0801  1792                  RSB                             ; Done
                        0802  1793                                                  ;
                        0802  1794 50$:             BUG_CHECK  INCONSTATE,FATAL     ; Queued state is inconsistent
                        0806  1795                                                  ;
                        0806  1796                  .DSABL  LSB
```

```
                     0806  1798 .SBTTL  'ACT_SYN_ERROR  - Synchronous block transfer error'
                     0806  1799 ;+
                     0806  1800 ;
                     0806  1801 ;  INPUTS:      R5      Scratch
                     0806  1802 ;               R4      ACB pointer
                     0806  1803 ;               R3      CSID of target system
                     0806  1804 ;               R2      CSD pointer
                     0806  1805 ;               R1      Scratch
                     0806  1806 ;               R0      Scratch
                     0806  1807 ;
                     0806  1808 ;  OUTPUTS:     R5      Garbage
                     0806  1809 ;               R4      ACB pointer
                     0806  1810 ;               R3      Garbage
                     0806  1811 ;               R2      Garbage
                     0806  1812 ;               R1      CEV$_EXIT
                     0806  1813 ;               R0      Low bit set to request state change
                     0806  1814 ;
                     0806  1815 ;-
                     0806  1816 ACT_SYN_ERROR:                                  ; Synchronous block transfer err
     20 A4    D4     0806  1817          CLRL    ACB$L_USER_AST(R4)            ; No AST delivery if synchronous
                     0809  1818                                                ; error return
56   3A A2    3C     0809  1819          MOVZWL  CSD$W_IOSB_STAT(R2),R6        ; Setup status to be returned
                     080D  1820                                                ; to EXE$CALL_CSP
   51   00    9A     080D  1821          MOVZBL  #CEV$_EXIT,R1                 ; No further events
   50   01    D0     0810  1822          MOVL    #1,R0                         ; Request state change
                05    0813  1823          RSB                                   ; Done
                     0814  1824
```

```
                     0814  1826 .SBTTL  'ACT_REQ_DEAL   - Illegal user deallocation request'
                     0814  1827 ;+
                     0814  1828 ;
                     0814  1829 ;  INPUTS:      R5      Scratch
                     0814  1830 ;               R4      ACB pointer
                     0814  1831 ;               R3      CSID of target system
                     0814  1832 ;               R2      CSD pointer
                     0814  1833 ;               R1      Scratch
                     0814  1834 ;               R0      Scratch
                     0814  1835 ;
                     0814  1836 ;  OUTPUTS:     R5      Garbage
                     0814  1837 ;               R4      0 to indicate CSD has been deallocated
                     0814  1838 ;               R3      Garbage
                     0814  1839 ;               R2      Garbage
                     0814  1840 ;               R1      CEVS_EXIT
                     0814  1841 ;               R0      Low bit clear to avoid state change
                     0814  1842 ;
                     0814  1843 ;-
                     0814  1844 ACT_REQ_DEAL:                                ; Illegal user dealloc. request
                     0814  1845 ;
                     0814  1846 ;
                     0814  1847 ;       The user has requested that the CSD be deallocated while the CSD
                     0814  1848 ;       is in the wrong state (e.g., a block transfer is in progress).
                     0814  1849 ;       Since this is a user error just prevent user AST notification and
                     0814  1850 ;       let the transfer run its course.  When the transfer completes and
                     0814  1851 ;       the 'special kernel' AST is delivered, return quotas and deallocate
                     0814  1852 ;       the CSD.
                     0814  1853 ;
                     0814  1854 ;       Note:
                     0814  1855 ;          This action routine could be rewritten to bug-check, but since
                     0814  1856 ;          since not all users have been updated yet to request AST
                     0814  1857 ;          notification, and since there is no adequate mechanism yet in
                     0814  1858 ;          place to detect image run-down (an interactive user may have
                     0814  1859 ;          Control-Y'd and issued a STOP) we do the next best thing:  stop
                     0814  1860 ;          the user AST delivery and return quota's when the operation
                     0814  1861 ;          actual completes.  The choice of when to return quota's is not
                     0814  1862 ;          perfect, but the choice was made since it may save the system
                     0814  1863 ;          from running out of pool at the expense of the process possibly
                     0814  1864 ;          running out of quota.
                     0814  1865 ;
                     0814  1866 ;          Eventually, each client must be updated to request AST
                     0814  1867 ;          notification even if it is not receiving any response.  Also, an
                     0814  1868 ;          image run-down hook is needed and a hook in ACKMSG to abort a
                     0814  1869 ;          transfer in progress.
                     0814  1870 ;
                     0814  1871 ;
  22 A2    D4        0814  1872 CLRL    CSD$A_ASTADR(R2)                    ; Prevent AST notification
  20 A4    D4        0817  1873 CLRL    ACB$L_USER_AST(R4)                  ; Here too
51   00    9A        081A  1874 MOVZBL  #CEV$_EXIT,R1                       ; No further events
50   01    D0        081D  1875 MOVL    #1,R0                               ; Allow state change
                05   0820  1876 RSB                                        ; Done
                     0821  1877
```

```
                                0821  1879 .SBTTL  'ACT_DEALL      - Deallocate CSD, return quotas'
                                0821  1880 ;+
                                0821  1881 ;
                                0821  1882 ;  INPUTS:       R5      Scratch
                                0821  1883 ;                 R4      ACB pointer
                                0821  1884 ;                 R3      CSID of target system
                                0821  1885 ;                 R2      CSD pointer
                                0821  1886 ;                 R1      Scratch
                                0821  1887 ;                 R0      Scratch
                                0821  1888 ;
                                0821  1889 ;  OUTPUTS:      R5      Garbage
                                0821  1890 ;                 R4      0 to indicate CSD has been deallocated
                                0821  1891 ;                 R3      Garbage
                                0821  1892 ;                 R2      Garbage
                                0821  1893 ;                 R1      CEV$_EXIT
                                0821  1894 ;                 R0      Low bit clear to avoid state change
                                0821  1895 ;
                                0821  1896 ;-
                                0821  1897 ACT_DEALL:                                       ; Deallocate CSD, return quota
      50    24 A4    3C       0821  1898          MOVZWL  ACB$L_USER_PID(R4),R0             ; Get process index
51    00000000'GF    D0       0825  1899          MOVL    G^SCH$GL_PCBVEC,R1                ; Get address of PCB vector
        50    6140   D0       082C  1900          MOVL    (R1)[R0],R0                       ; Get PCB itself
   60 A0    24 A4    D1       0830  1901          CMPL    ACB$L_USER_PID(R4),PCB$L_PID(R0); Is this process still here?
             0D       12       0835  1902          BNEQ    DEALL_CSD                         ; If NEQ, no
                               0837  1903                                                    ;
      51    08 A4    3C       0837  1904          MOVZWL  ACB$W_SIZE(R4),R1                 ; Get quota taken
   50    0080 C0    D0       083B  1905          MOVL    PCB$L_JIB(R0),R0                  ; Get JIB
     20 A0    51    C0       0840  1906          ADDL    R1,JIB$L_BYTCNT(R0)               ; Return quota
                               0844  1907                                                    ;
                               0844  1908 DEALL_CSD:                                       ; Deallocate CSD/ACB
17 31 A4     04    E5       0844  1909          BBCC    #ACB$V_STS_PCNT,ACB$B_STS(R4),30$; If BC, not part of Bcst count
   50    2C A4    D0       0849  1910          MOVL    ACB$L_PARENT(R4),R0               ; Get parent ACB, if any
         2C A4    D4       084D  1911          CLRL    ACB$L_PARENT(R4)                  ; Erase pointer
   02    0A A0    91       0850  1912          CMPB    ACB$B_TYPE(R0),#DYN$C_ACB         ; Check packet type
             27    12       0854  1913          BNEQ    200$                              ; If NEQ, pool corruption
         28 A0    B7       0856  1914          DECW    ACB$W_WAIT_CNT(R0)                ; Decrement the wait count
             05    12       0859  1915          BNEQ    30$                               ; If NEQ, not done yet
00 31 A0     02    E5       085B  1916          BBCC    #ACB$V_STS_WAIT,ACB$B_STS(R0),30$ ; If BC, not waiting
      50     54    D0       0860  1917 30$:     MOVL    R4,R0                             ; Get address for deallocation
             54    D4       0863  1918          CLRL    R4                                ; Erase offical pointer
   02    0A A0    91       0865  1919          CMPB    ACB$B_TYPE(R0),#DYN$C_ACB         ; Check packet type
             12    12       0869  1920          BNEQ    200$                              ; If NEQ, pool corruption
         28 A0    B5       086B  1921          TSTW    ACB$W_WAIT_CNT(R0)                ; Any lingering references?
             11    12       086E  1922          BNEQ    210$                              ; If NEQ yes, bug
 00000000'GF      16       0870  1923          JSB     G^EXE$DEANONPAGED                 ; Deallocate the block
                               0876  1924                                                    ;
      50    01    D0       0876  1925          MOVL    S^#SS$_NORMAL,R0                  ; Why not
      51    00    9A       0879  1926          MOVZBL  #CEV$_EXIT,R1                     ; No further events
                  05       087C  1927          RSB                                       ; Done
                               087D  1928                                                    ;
                               087D  1929 200$:    BUG_CHECK  INCONSTATE,FATAL           ; ACB$B_TYPE is wrong
                               0881  1930 210$:    BUG_CHECK  INCONSTATE,FATAL           ; WAIT_CNT non-zero
                               0885  1931
```

```
                     0885  1933 .SBTTL  'ACT_BUG        - Bugcheck failure'
                     0885  1934 .SBTTL  'ACT_NYI        - Not-yet-implemented error'
                     0885  1935 .SBTTL  'ACT_NOP        - No-operation'
                     0885  1936 ;+
                     0885  1937 ;
                     0885  1938 ;
                     0885  1939 ;    INPUTS:     R5      ACB ptr or zero
                     0885  1940 ;
                     0885  1941 ;    OUTPUTS:    R5      Unchanged
                     0885  1942 ;
                     0885  1943 ;
                     0885  1944 ;-
                     0885  1945 ACT_BUG:
                     0885  1946         BUG_CHECK  INCONSTATE,FATAL     ; Signal the bug
                     0889  1947 ACT_NYI:
                     0889  1948         BUG_CHECK  INCONSTATE,FATAL     ; Signal the bug
                     088D  1949
                     088D  1950 ACT_NOP:                                ; Nop action routine
51   00   D0           088D  1951         MOVL    S^#CEV$_EXIT,R1        ; No further events
50   01   90           0890  1952         MOVB    #1,R0                  ; Allow state transition
                05      0893  1953         RSB
                        0894  1954
                        0894  1955
                        0894  1956 .END
```

```
$$BASE                  =  00000002
$$DISPL                 =  00000007
$$GENSW                 =  00000001
$$HIGH                  =  00000006
$$LIMIT                 =  00000004
$$LOW                   =  00000002
$$MNSW                  =  00000001
$$MXSW                  =  00000001
ACB$B_RMOD              =  0000000B
ACB$B_STA                  00000030
ACB$B_STS                  00000031
ACB$B_TYPE              =  0000000A
ACB$K_CSPLNG            =  00000034
ACB$K_LENGTH            =  0000001C
ACB$K_RETRY             =  00000004
ACB$L_AST               =  00000010
ACB$L_ASTPRM            =  00000014
ACB$L_KAST              =  00000018
ACB$L_PARENT               0000002C
ACB$L_PID               =  0000000C
ACB$L_USER_AST             00000020
ACB$L_USER_PID             00000024
ACB$M_KAST              =  00000080
ACB$M_NODELETE          =  00000020
ACB$M_STS_ASY           =  00000001
ACB$M_STS_BCST          =  00000008
ACB$M_STS_PCNT          =  00000010
ACB$M_STS_WAIT          =  00000004
ACB$V_STS_ASY           =  00000000
ACB$V_STS_BCST          =  00000003
ACB$V_STS_PCNT          =  00000004
ACB$V_STS_QUE           =  00000001
ACB$V_STS_WAIT          =  00000002
ACB$W_LAST_INX             0000002A
ACB$W_RETRY                00000032
ACB$W_SIZE              =  00000008
ACB$W_WAIT_CNT             00000028
ACT_BLOCK_XFER             0000073F  R     02
ACT_BUG                    00000885  R     02
ACT_DEALL                  00000821  R     02
ACT_FORK_WAIT              000006F4  R     02
ACT_GET_CDRP               00000699  R     02
ACT_GIVE_UP                000007CC  R     02
ACT_INSQUE                 00000677  R     02
ACT_NOP                    0000088D  R     02
ACT_NO_AST                 000007BE  R     02
ACT_NYI                    00000889  R     02
ACT_QUE_AST                000007E6  R     02
ACT_QUE_KAST               000007D7  R     02
ACT_REMQUE                 00000689  R     02
ACT_REQ_DEAL               00000814  R     02
ACT_REQ_ILL_BT             00000731  R     02
ACT_SYN_ERROR              00000806  R     02
AST                        000005CE  R     02
ASTEVT                     000005E7  R     02
BIT...                  =  00000003
BUG$_INCONSTATE            ********    X   02
CDRP$B_CLTSTS           =  0000004B
CDRP$B_CNXRMOD          =  0000004A
CDRP$K_CM_LENGTH        =  00000060
CDRP$L_CNXBCNT          =  00000046
CDRP$L_CNXSVAPTE        =  00000040
CDRP$L_CSP_CSD          =  00000060
CDRP$L_CSP_SP1          =  00000064
CDRP$L_LBOFF            =  00000030
CDRP$L_MSGBLD           =  0000004C
CDRP$L_RBOFF            =  00000038
CDRP$L_VAL2             =  00000030
CDRP$L_VAL5             =  0000003C
CDRP$L_XCT_LEN          =  0000003C
CDRP$M_CSP_ERROR        =  00000001
CDRP$M_CSP_FLWCTL       =  00000004
CDRP$M_CSP_QUEUED       =  00000002
CDRP$V_CSP_ERROR        =  00000000
CDRP$V_CSP_FLWCTL       =  00000002
CDRP$V_CSP_QUEUED       =  00000001
CDRP$W_CNXBOFF          =  00000044
CEV$AB_RSP_CEV             00000154  R     02
CEV$AL_ACTTAB              00000000  R     02
CEV$AW_STA_TAB             00000094  R     02
CEV$K_STATES            =  00000006
CEV$K_STA_.             =  00000005
CEV$K_STA_A             =  00000004
CEV$K_STA_F             =  00000001
CEV$K_STA_I             =  00000000
CEV$K_STA_K             =  00000003
CEV$K_STA_S             =  00000005
CEV$K_STA_X             =  00000002
CEV$_AST_DEL            =  0000000D
CEV$_BT_DONE            =  00000007
CEV$_BT_SYNERR          =  00000008
CEV$_BUG                =  00000001
CEV$_CSP_BUSY           =  00000009
CEV$_EXIT               =  00000000
CEV$_FORK_DONE          =  00000005
CEV$_GIVE_UP            =  0000000B
CEV$_GOT_CDRP           =  00000006
CEV$_INV_PID            =  0000000F
CEV$_KAST_DEL           =  0000000C
CEV$_MAX_EVT            =  0000000F
CEV$_NO_AST             =  0000000E
CEV$_NO_CDRP            =  00000004
CEV$_NO_CSP             =  0000000A
CEV$_REQ_BT             =  00000002
CEV$_REQ_DEALL          =  00000003
CLEAN_UP                   000001A3  R     02
CLEAN_UP1                  000001A7  R     02
CLMHDR$K_BT_LENGTH      =  00000018
CLSMSG$B_FACILITY       =  00000008
CLSMSG$B_FUNC           =  00000009
CLSMSG$K_FAC_CSP        =  00000006
CLSMSG$M_RESPMSG        =  00000080
CLU$GL_CLUB                ********    X   02
CLU$GL_CLUSVEC             ********    X   02
```

```
CLU$GW_MAXINDEX                 ********    X   02
CLUB$L_CSPBL                  = 0000008C
CLUB$L_CSPFL                  = 00000088
CLUB$L_CSPIPID                = 00000090
CLUB$L_LOCAL_CSB              = 00000010
CNX$ALLOC_WARMCDRP              ********    X   02
CNX$BLOCK_READ                  ********    X   02
CNX$BLOCK_WRITE                 ********    X   02
CNX$BLOCK_XFER                  ********    X   02
CNX$DEALL_WARMCDRP_CSB          ********    X   02
CNX$PARTNER_INIT_CSB            ********    X   02
CNX$PARTNER_RESPOND             ********    X   02
COMMON_SETUP                    00000574 R      02
CSB$L_CSID                    = 0000004C
CSD$AB_DATA                   = 00000052
CSD$A_ASTADR                  = 00000022
CSD$B_SUBTYPE                 = 0000000B
CSD$B_TYPE                    = 0000000A
CSD$K_LENGTH                  = 00000052
CSD$L_CSID                    = 0000000E
CSD$L_IMGCNT                  = 0000004E
CSD$L_IPID                    = 00000036
CSD$L_PROCUIC                 = 0000004A
CSD$L_RECVLEN                 = 0000001A
CSD$L_RECVOFF                 = 0000001E
CSD$L_SENDLEN                 = 00000012
CSD$L_SENDOFF                 = 00000016
CSD$Q_INT_IOSB                = 0000003A
CSD$Q_PROCPRIV                = 00000042
CSD$W_CODE                    = 0000000C
CSD$W_IOSB_STAT               = 0000003A
CSD$W_SIZE                    = 00000008
CSP$BEGIN                       00000000 RG     02
CSP$B_INITED                    00000171 R      02
CSP$B_RCVCSDCNT                 00000170 R      02
CSP$DISPATCH                    000001CD RG     02
CSP$INIT                        00000172 RG     02
CSP$K_MAX_FLWCTL              = 00000008
CSP$Q_ACB_IDLE                  00000160 R      02
CSP$Q_ACB_XFER                  00000168 R      02
CSP$_ABORT                    = 00000002
CSP$_BADCSD                   = 00000003
CSP$_DONE                     = 00000004
CSP$_LOCAL                    = 00000007
CSP$_REJECT                   = 00000006
CSP$_REPLY                    = 00000005
CSPMSG$B_RSP                    00000018
CSPMSG$B_SPARE                  00000019
CSPMSG$K_RSP_ASYNERR          = 00000007
CSPMSG$K_RSP_BADCSD           = 00000006
CSPMSG$K_RSP_BUSY             = 00000002
CSPMSG$K_RSP_ILL              = 00000001
CSPMSG$K_RSP_MAX              = 00000009
CSPMSG$K_RSP_NOCSP            = 00000003
CSPMSG$K_RSP_NOP              = 00000000
CSPMSG$K_RSP_RO               = 00000004
CSPMSG$K_RSP_RW               = 00000005
CSPMSG$K_RSP_SYNERR           = 00000008
CSPMSG$L_CSD_SIZE               0000001C
CSPMSG$W_CLIENT                 0000001A
CSP_COMMAND                     00000308 R      02
CSP_COMMAND_1                   000002F0 R      02
CTL$GL_PCB                      ********      X 02
CTL$GL_PHD                      ********      X 02
DEALL_CSD                       00000844 R      02
DUMP_CDRP                       000007A8 R      02
DYN$C_ACB                     = 00000002
DYN$C_CLU                     = 00000065
DYN$C_CSD                     = 00000064
EXE$ALLOCBUF                    ********      X 02
EXE$ALLOC_CSD                   00000435 RG     02
EXE$ALONONPAGED                 ********      X 02
EXE$BUFQUOPRC                   ********      X 02
EXE$CSP_BRDCST                  00000357 RG     02
EXE$CSP_CALL                    0000051E RG     02
EXE$CSP_COMMAND                 0000028E RG     02
EXE$DEALLOC_CSD                 0000050C RG     02
EXE$DEANONPAGED                 ********      X 02
EXE$FORK_WAIT                   ********      X 02
FKB$B_FIPL                    = 0000000B
FKB$L_FPC                     = 0000000C
FKB$L_FR3                     = 00000010
FKB$L_FR4                     = 00000014
GET_NEXT_CSB                    000003E3 R      02
INSQUE_CLUB                     0000025C R      02
IPL$_ASTDEL                   = 00000002
IPL$_SCS                      = 00000008
IPL$_SYNCH                    = 00000008
JIB$C_BYTCNT                  = 00000020
KAST                            000005C4 R      02
MMG$GL_SPTBASE                  ********    X   02
PCB$L_JIB                     = 00000080
PCB$L_PID                     = 00000060
PCB$L_STS                     = 00000024
PCB$L_UIC                     = 000000BC
PCB$Q_PRIV                    = 00000084
PCB$V_SSRWAIT                 = 0000000A
PHD$L_IMGCNT                  = 000000F4
PR$_IPL                         ********    X   02
PRI$_IOCOM                    = 00000001
PROC_EVENT                      00000628 R      02
PROC_EVENT_ASY                  0000060F R      02
REQ_MSGBLD                      000006DA R      02
RSN$_ASTWAIT                  = 00000001
RSN$_NPDYNMEM                 = 00000003
RSP_MSGBLD                      00000349 R      02
SCH$GL_PCBVEC                   ********    X   02
SCH$QAST                        ********    X   02
SCH$RWAIT                       ********    X   02
SCH$WAKE                        ********    X   02
SIZ...                        = 00000001
SS$_BADPARAM                  = 00000014
SS$_DEVACTIVE                 = 000002C4
SS$_NORMAL                    = 00000001
```

```
SS$_NOSUCHNODE                  = 0000028C
SS$_REJECT                      = 00000294
SS$_TIMEOUT                     = 0000022C
VA$M_BYTE                       = 000001FF
VA$S_VPN                        = 00000015
VA$V_VPN                        = 00000009
WAIT                              0000054D R      02
_$END                           = 0000015E R      02
_$ENT                           = 00000002
_$MAXINX                        = 00000024
_$START                         = 00000154 R      02
_$TMP                           = 00000000 R      02
```

Psect synopsis

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
ABS	00000034 (52.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
$$$200	00000894 (2196.)	02 (2.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC QUAD

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	36	00:00:00.05	00:00:01.28
Command processing	137	00:00:00.48	00:00:04.21
Pass 1	556	00:00:16.55	00:00:54.85
Symbol table sort	0	00:00:02.15	00:00:08.44
Pass 2	338	00:00:04.20	00:00:12.97
Symbol table output	29	00:00:00.13	00:00:00.98
Psect synopsis output	1	00:00:00.02	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	1099	00:00:23.58	00:01:22.75

The working set limit was 2400 pages.
141751 bytes (277 pages) of virtual memory were used to buffer the intermediate code.
There were 110 pages of symbol table space allocated to hold 1983 non-local and 75 local symbols.
1956 source lines were read in Pass 1, producing 21 object records in Pass 2.
48 pages of virtual memory were used to define 46 macros.

Macro library statistics

Macro library name	Macros defined
_255DUA28:[SYSLOA.OBJ]CLUSTER.MLB;1	4
_255DUA28:[SYS.OBJ]LIB.MLB;1	21
_255DUA28:[SYSLIB]STARLET.MLB;2	7
TOTALS (all libraries)	32

2066 GETS were required to define 32 macros.

```
There were no errors, warnings or information messages.

MACRO/LIS=LIS$:CSPCALL/OBJ=OBJ$:CSPCALL MSRC$:CSPCALL/UPDATE=(ENH$:CSPCALL)+EXECML$/LIB+LIB$:CLUSTER/LIB
```

CSPCALL
LIS

CSPCJFMAS
LIS

CSP
LIS

CSPBRKTHR
LIS

CONUTIL
LIS

CSPCALLACT
LIS

CONSUBS
LIS